طراحی الگوریتم- تقسیم و غلبه

محسن هوشمند

تقسیم و غلبه

تابعی با n ورودی داریم. روش تقسیم و غلبه بر افراز آن به k زیرمجموعه متمایز سعی دارد و مسئله اصلی بزرگ را به k زیرمسئله کوچکتر تقسیم می‌کند. سپس زیرمسائل ایجاد شده حل و با اسلوبی از ترکیب جواب مسئلهٔ اصلی حاصل خواهد شد. ممکن است تقسیم صرفا در یک مرحله انجام نگیرد و خود زیرمسائل به زیرمسائلی کوچکتر تقسیم شوند. زیرمسائل غالبا هم‌نوع مسئلهٔ اصلی هستند. بنابراین، الگوریتم‌های بازگشتی را در صورت‌بندی و حل آنها می‌توان تعریف کرد و بکار برد. دید بازگشتی موجب می‌شود مسئله را به صورت تکراری از نوع بازگشت به مسائل کوچک و کوچکتر تقسیم نمود تا زمانی که حل آنها بدون تقسیم بیشتر ممکن شود.

سخن کوتاه، چنین بازنمایشی چارچوبی انتزاعی از روش تقسیم و غلبه را با آرایهٔ فرضی ورودی A[۱, n] به صورت زیر به دست می‌دهد.

```
Algorithm ToQ(chap, rast)
  If kam(chap, rast):
     then return qalabe(chap, rast)
  else:
    vasat = taqsim(chap, rast)
    return tarkib( ToQ(chap, vasat), ToQ(vasat+1, rast)
```

تابع kam تابعی با خروجی دو مقداری است که کافی بودن کوچکی مسئله را معین می‌کند. اگر تابع به اندازهٔ کافی کوچک باشد، تابع اجرای qalabe فراخوانی می‌شود. درغیر این‌صورت مسئله به دو مسئلهٔ کوچکتر تقسیم و هر یک به صورت بازگشتی حل می‌شوند.

اگر زمان اجرای بخش ترکیب ToQ را با f(n) نمایش دهیم، آن‌گاه می‌توان با الگوریتم تقسیم و غلبه زمان را به صورت زیر تعریف کرد

$$z(n) = \begin{cases} z\left(\left\lceil \frac{n}{۲} \right\rceil\right) + z\left(\left\lfloor \frac{n}{۲} \right\rfloor\right) + f(n), n\text{بزرگ} \\ g(n), n\text{کوچک} \end{cases}$$

اگر $n = ۲^{m}$ آن‌گاه معادلهٔ بالا به صورت زیر ساده‌تر می‌شود.

$$z(n) = \begin{cases} ۲z\left(\frac{n}{۲}\right) + f(n), n\text{بزرگ} \\ g(n), n\text{کوچک} \end{cases}$$

جستجوی دودوئی

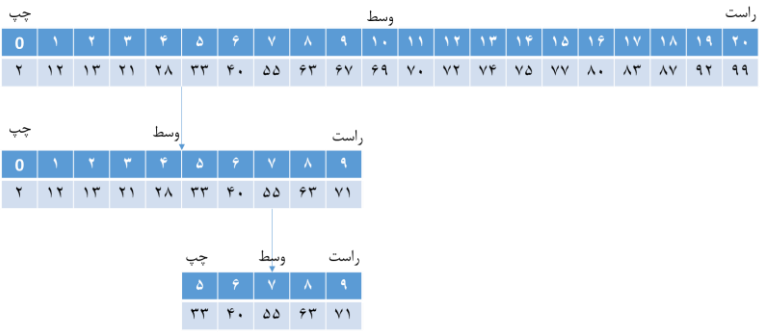
گاهی اوقات دنبال یافتن مقداری هستیم. به فرایند مذکور «جستجو» گویند. در فصل قبل الگوریتم‌های جستجو و جستجو دودوئی بررسی شد.

قبلا در بعضی از مسائل دیدیم که می‌توان کارایی زمانی الگوریتم‌ها را بهبود داد. در الگوریتم جستجوی ساده در بدترین حالت به تعداد ورودی‌ها جستجو انجام خواهد شد. اما اگر فهرست مرتب باشد و مقادیر از کوچک به بزرگ مرتب باشند می‌توان بدترین حالت را به میزان زیادی کاهش داد. یکی از الگوریتم‌های مهم جستجو که دارای بدترین زمان جستجوی کمتری باشد «الگوریتم جستجو دودوئی» است. در الگوریتم مذکور در ابتدا تعداد فهرست مشخص و سپس بر اساس آن اندیس عدد میانه را تعیین می‌کند. مقدار کلید با عدد متناظر اندیس میانه مقایسه می‌شود اگر برابر باشند، اندیس برگردانده می‌شود و اگر کوچکتر باشد در نیمه چپ فهرست و اگر کلید بزرگتر باشد در نیمهٔ راست فهرست جستجو ادامه خواهد یافت. عمل مذکور در هر نیمه ادامه خواهد یافت. در واقع هر دفعه جستجو در نیمی از فهرست ادامه می‌یابد. می‌توان نشان داد که در بدترین حالت لگاریتمی در مبنای دو مقایسه انجام خواهد پذیرفت. به عنوان مثال اگر طول فهرست ۱۰۰۰ باشد، در جستجوی معمولی در بدترین حالت ۱۰۰۰ مقایسه، ولی در جستجوی درختی حداکثر ۱۰ مقایسه انجام خواهد یافت که بهبودی بسیار مناسب است.

پس در ابتدا دو اندیس چپ و راست که معادل اندیس اولین مقدار و آخرین مقدار فهرستند را تعریف می‌کنیم. تا زمانی که چپ‌ترین اندیس از اندیس راست کوچکتر یا مساوی باشد حلقه را ادامه می‌دهیم. در هر تکرار حلقهٔ اندیس میانی برابر میانگین جز صحیح دو اندیس قبلی تعریف می‌شود. اگر کلید بزرگتر از مقدار متناظر اندیس میانی باشد، پرچم اندیس چپ برابر با مقدار بعدی اندیس میانی قرار می‌گیرد و اگر کلید کوچکتر باشد مقدار اندیس راست برابر با اندیس قبل از اندیس میانی قرار می‌گیرد. در غیر این صورت مقدار کلید و مقدار متناظر اندیس میانی برابرند، پس اندیس میانی برگشت داده می‌شود. در صورتی حلقه به پایان برسد و اندیسی در حلقه برگردانده نشود به معنای نبود مقدار کلید در فهرست است.

در مثال قبل دیدیم که فهرست مرتب چه تاثیر بزرگی بر بهینگی الگوریتم جستجو می‌گذارد. بنابراین سعی می‌شود که ساختار داده‌های چند مقداری مرتب باشند تا بتوان جستجو دودوئی را روی آنها اعمال کرد. در نتیجه سعی بر مرتب‌سازی چند مقداری‌ها می‌شود. همچنین مرتب‌سازی تاثیر فراوان در دانش و فن رایانه داشته است. در ادامه چند نوع الگوریتم مرتب‌سازی معرفی می‌شوند. تفاوت آنها در میزان استفاده بهینه منابع سیستم شامل حافظه و زمان است.

تحلیل زمانی



تمرین- جستجو دودوئی چه قرابت و تفاوتی با چارچوب تقسیم و غلبه دارد؟ در جستجو دودوئی مسئله به زیرمسئله‌ها شکسته می‌شود، اما بخش ترکیب پاسخ‌ها وجود ندارد. در واقع، به همین دلیل نیاز به تعریفی بازگشتی نیست و مسئله را می‌توان به صورت تکراری حل کرد و درنتیجه اجرای الگوریتم را به دلیل دوری از پشته می‌توان سریعتر پیش برد.

فرض می‌کنیم که n توانی از ۲ است. بدترین زمان؟ مقدار کلید بزرگتر از مقادیر فهرست باشد. آن‌گاه

$$B(n) = B\left(\frac{n}{۲}\right) + ۱, n > ۲$$

$$B\left(۱\right) = ۱$$

با توجه فصل «تحلیل الگوریتم» داریم

$$B(n) = \log n + ۱$$

اگر n توانی از دو نباشد داریم

$$B(n) = \lfloor \log n \rfloor + ۱ = \theta(\log n)$$

تمرین- اثبات کنید.

بزرگ‌ترین و کوچکترین اعضای آرایه

گاهی اوقات به دنبال یافتن کوچکترین و بزرگترین اعضای آرایهٔ ورودی هستیم. اولین الگوریتمی که به نظر می‌رسد، اولین عضو را برابر با کوچکترین و بزرگترین قرار می‌دهیم. هر عضو بعدی را بررسی می‌کنیم. در صورتی که کوچکتر باشد، با کوچکترین عضو و در صورتی که بزرگتر از باشد به عنوان بزرگترین عضو انتخاب می‌شود. این روند تا پایان آرایه ادامه می‌یابد. روش‌سادهٔ مذکور به صورت زیر پیاده می‌شود.

```
#include <stdio.h>
struct JoftBK {
   int bish;
   int kam;
};
// Tabe yaftan-e bish o kam
struct JoftBK bishkam_sade(int arr[], int n) {
   struct JoftBK ntije;
   ntije.bish = arr[0];
   ntije.kam = arr[0];
   // halqe jahat yaftan bish o kam-e araye
   for (int i = 1; i < n; i++) {
      if (arr[i] > ntije.bish) {
         ntije.bish = arr[i]; // broz karadane mqdare bish dar sorate yaftan adad bozorgtar
      }
      if (arr[i] < ntije.kam) {
         ntije.kam = arr[i]; // broz karadane mqdare kam dar sorate yaftan adad kochektar
      }
   }
   return ntije; // JoftBK havi-e mqadire bish o kam
}
int main() {
   int arr[] = {16, 12, -12, 4, 26, 54, 34, 64, 77, -13};
   int n = sizeof(arr) / sizeof(arr[0]);
   //Farakhani Tabe
            struct JoftBK ntije = bishkam_sade(arr, n);
   printf("Ozv bish-e araye: %d\n", ntije.bish);
   printf("Ozv kam-e araye: %d\n", ntije.kam);
   return 0;
}
```

پس $n - 1$ مقایسه برای یافتن بیش و $n - 1$ مقایسه برای یافتن کم و در نتیجه در کل $2n - 2$ مقایسه لازم است. هر چند می‌توان با تعداد مقایسه‌های کمتری هر دو مقدار کم و بیش را حل کرد. فرض مجموعه داده را به مجموعه‌های دو عضوی تقسیم می‌کنیم. پس $\frac{n}{2}$ مجموعه خواهیم داشت. حال بزرگ و کوچک هر مجموعه دو عضوی را بدست می‌آوریم. هر زیرمجموعه به یک مقایسه نیاز دارد پس جمعا $\frac{n}{2}$ مقایسه جهت کم و بیش هر مجموعه دو عضوی لازم است. پس تمامی بزرگ‌ها را با یکدیگر جهت یافتن بزگترین عدد کل مجموعه و تمامی کوچک‌ها را با یکدیگر جهت یافتن کوچکترین مقایسه

می‌کنیم. یافتن بیشترین نیاز به مقایسه و یافتن کمترین نیز به $\frac{n}{2} - 1$ مقایسه نیاز دارد. در نتیجه در کل $\frac{3n}{2} - 2$ مقایسه انجام خواهد یافت.

روش تقسیم و غلبه: آرایه ورودی را بازگشتانه به دو زیرآرایهٔ تقسیم، و بیش و کم هر یک را بررسی می‌کنیم. بیشینه از مقایسه بیشینه‌های هر دو زیرآرایه و کمینه از مقایسهٔ کمینه‌های دو زیرآرایه یافت خواهد شد. کد سی روش تقسیم و غلبه به صورت زیر است.

```
#include <stdio.h>
struct JoftBK {
   int bish;
   int kam;
};
// Tabe yaftan e bish o kam
struct JoftBK bishkam_ToQ(int arr[], int paeen, int bala) {
   struct JoftBK ntije;
                struct JoftBK chap;
                struct JoftBK rast;
                int vst;
                // vojode yek ozv dar araye
                if ( paeen == bala){
                                ntije.bish = arr[paeen];
                                ntije.kam = arr[paeen];
                                return ntije;
                }


                // vojode do ozv dar araye
                if ( paeen == bala - 1){
                                if (arr[paeen] < arr[bala]){
                                                ntije.bish = arr[bala];
                                                ntije.kam = arr[paeen];
                                }
                                else{
                                                ntije.bish = arr[paeen];
                                                ntije.kam = arr[bala];
                                }


                                return ntije;
                }


                // vojode bish az do ozv dar araye
                vst = (paeen + bala) / 2;
                chap = bishkam_ToQ(arr, paeen, vst);
                rast = bishkam_ToQ(arr, vst + 1, bala);
   // mqayese
                ntije.bish = (chap.bish > rast.bish) ? chap.bish: rast.bish;
```

```
    ntije.kam = (chap.kam < rast.kam) ? chap.kam: rast.kam;


    return ntije;
}
int main() {
    int arr[] = {16, 12, -12, 4, 26, 54, 34, 64, 77, -13};
    int n = sizeof(arr) / sizeof(arr[0]);
    struct JoftBK ntije = bishkam_ToQ(arr, 0 , n - 1);
    printf("Ozv bish-e araye: %d\n", ntije.bish);
    printf("Ozv kam-e araye: %d\n", ntije.kam);
    return 0;
}
```

تحلیل زمانی الگوریتم تقسیم و غلبهٔ معرفی شده جهت یافتن کم و بیش آرایه به صورت زیر است.

$$z(n) = \begin{cases} z\left(\left\lfloor \frac{n}{۲} \right\rfloor\right) + z\left(\left\lceil \frac{n}{۲} \right\rceil\right) + ۲, n > ۲ \\ ۱ \qquad\qquad\qquad\qquad , n = ۲ \\ 0 \qquad\qquad\qquad\qquad , n = ۱ \end{cases}$$

که در صورت بررسی می‌توان دید که برابر با مقدار $۲ - ۳\left\lceil \frac{n}{۲} \right\rceil$ است. تمرین- اثبات کنید.

فرض n توانی از ۲ باشد. داریم

$$z(n) = \begin{cases} ۲z\left(\frac{n}{۲}\right) + ۲ \qquad\qquad , n > ۲ \\ ۱ \qquad\qquad\qquad\qquad , n = ۲ \\ 0 \qquad\qquad\qquad\qquad , n = ۱ \end{cases}$$

$$z(n) = ۲z\left(\frac{n}{۲}\right) + ۲ = ۲\left[۲z\left(\frac{n}{۲^{۲}}\right) + ۲\right] + ۲ = ۲ + ۲^{۲} + ۲^{۲}z\left(\frac{n}{۲^{۲}}\right)$$

$$= ۲ + ۲^{۲} + ۲^{۲}\left[۲z\left(\frac{n}{۲^{۳}}\right) + ۲\right] = ۲ + ۲^{۲} + ۲^{۳} + ۲^{۳}z\left(\frac{n}{۲^{۳}}\right)$$

$$\vdots$$

$$= ۲ + ۲^{۲} + ۲^{۳} + \cdots + ۲^{m-۱} + ۲^{m-۱} z\left(۲\right)$$

$$= ۲^{m} - ۲ + ۲^{m-۱} = n - ۲ + \frac{n}{۲} = \frac{۳}{۲}n - ۲$$

تمرین-نشان دهید که راه‌حل بالا همیشه بهینه نیست. راه حل بهینه‌تری را معرفی کنید.

سخن کوتاه اگر زمان n توانی از ۲ باشد، آن‌گاه $\frac{۳}{۲}n - ۲$ مقایسه در بهترین حالت و $\left(n - ۱\right)$ ۲ مقایسه در بدترین حالت

هر الگوریتم مبتنی بر مقایسه دست کم $\frac{۳}{۲}n - ۲$ مقایسه لازم دارد. اما الگوریتم تقسیم و غلبهٔ کم-بیش لزوما بهینه نیست. دلیل آن استفاده از فضای زیاد برای ذخیره انواع متغیرهاست. و برای آرایهٔ n عضوی نیاز به $\left\lfloor \log_{۲} n \right\rfloor + ۱$ سطح فراخوانی دارد و هر فراوانی نیز به تعداد ۹ مقدار ذخیره می‌شوند.

ضرب چندجمله‌ای‌ها

دو چند جمله‌ای مانند جمله‌ای‌های زیر را در نظر می‌گیریم.

$$p(x) = a_{۰} + \cdots + a_{n-۲}x^{n-۲} + a_{n-۱}x^{n-۱}$$

$$q(x) = b_{۰} + \cdots + b_{n-۲}x^{n-۲} + b_{n-۱}x^{n-۱}$$

هدف محاسبهٔ $r(x) = p(x)q(x)$ است. با توجه به بزرگترین درجه p و q بزرگترین درجه r برابر با $۲n - ۲$ خواهد بود و داریم:

$$r(x) = c_{۰} + \cdots + c_{n-۲}x^{۲n-۳} + c_{n-۱}x^{۲n-۲}$$

و c_i برابر است با

$$c_i = \sum_{i=۰}^{k} a_i b_{k-i} = \sum_{i+j=k} a_i b_j \, , k = ۰,۱, \ldots , ۲n - ۲$$

$$c_{۰} = a_{۰}b_{۰}$$

$$c_1 = a_0 b_1 + a_1 b_0$$

تمرین- چرا بزرگترین توان برابر $2n - 2$ است؟ اندازهٔ طول آرایهٔ حاوی نتیجه چقدر است؟

شبه کد ضرب دو چندجمله‌ای به صورت زیر است.

```
Alg. Zarb_Chandjomleye(A, B)
C[0:2n-2] = 0
for i = 0 : n - 1
   for j = 0 : n – 1
      C[i + j] = C[i + j] + A[i] * B[j]
```

ضرایب را می‌توان با فهرست نمایش داد. برنامه زیر نمایش محاسبهٔ ضرائب را به صورت ساده انجام می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>


// Tabe zarb do chandjomleye
void zarbChanjomlye(int *C1, int *C2, int *hasel, int m, int n) {
   // mqdardehi avalie
   for (int i = 0; i < m + n - 1; i++) {
      hasel[i] = 0;
   }


   // zarb zarayeb motanezer
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         hasel[i + j] += C1[i] * C2[j];
      }
   }
}


int main() {
   int C1[] = {3, 1, 0, 1, 10, 6}; // 3 + 1x + 0x^2 + 10x^3 + 6x^4
   int C2[] = {4, 2, 1};    // 4 + 2x + 1x^2
   int m = sizeof(C1) / sizeof(C1[0]);
   int n = sizeof(C2) / sizeof(C2[0]);


   // chandjomleye haseldaraye darajeye m + n - 2
   int *hasel = (int *)malloc((m + n - 1) * sizeof(int));


   // zarb-e chandjomleye-ha the polynomials
   zarbChanjomlye(C1, C2, hasel, m, n);
```

```
    // Nemayesh hasel
    printf("Chanjomleye hasel: ");
    for (int i = 0; i < m + n - 1; i++) {
        printf("%d ", hasel[i]);
    }
    printf("\n")
    // rahasazi hafeze
    free(hasel);
    return 0;
}
```

تمرین- چرا در الگوریتم معرفی شده نحوهٔ محاسبهٔ C[i + j] = C[i + j] + A[i] * B[j] درست است؟ به دیگر سخن، معادله چگونه با دو حلقه معادل $c_i = \sum_{i=0}^{k} a_i b_{k-i}$ است.

مرتبهٔ زمانی الگوریتم معرفی شده برابر با $O\left(n^{۲}\right)$ یا به طور دقیقتر $\theta\left(n^{۲}\right)$ است.

تقسیم و حل- تلاش نخست

در ادامه سعی می‌کنیم ضرب دوچندجمله‌ای را از طریق تقسیم و غلبه حل کنیم. به عبارت دیگر، هر چندجمله‌ای را همانند مثال زیر به دو بخش چپ و راست تقسیم می‌کنیم.

$$p(x) = ۱ + x + ۳x^{۲} - ۴x^{۳} = ۱ + x + \left(۳ - ۴x\right)x^{۲} = p_c(x) + x^{۲}p_r(x)$$

$$q(x) = ۱ + ۲x - ۵x^{۲} - ۳x^{۳} = ۱ + ۲x + (-۵ - ۳x)x^{۲} = q_c(x) + x^{۲}q_r(x)$$

حال، محاسبهٔ $r(x)$ را به صورت زیر بازنویسی می‌کنیم.

$$r(x) = p(x)q(x) = \left(p_c(x) + x^{۲}p_r(x)\right)\left(q_c(x) + x^{۲}q_r(x)\right)$$

$$= p_c(x)q_c(x) + x^{۲}[p_r(x)q_c(x) + p_c(x)q_r(x)] + x^{۴}p_r(x)\ q_r(x)$$

به بیان دیگر، در حالت کلی داریم:

$$r(x) = p(x)q(x) = \left(p_c(x) + x^{\frac{n}{۲}}p_r(x)\right)\left(q_c(x) + x^{\frac{n}{۲}}q_r(x)\right)$$

$$= p_c(x)q_c(x) + x^{\frac{n}{۲}}[p_r(x)q_c(x) + p_c(x)q_r(x)] + x^{n}p_r(x)\ q_r(x).$$

سخن کوتاه، حاصلضرب دو چندجمله‌ای را با تقسیم چنجمله‌ای‌ها به چندجمله‌ای‌های کوچکتر انجام می‌دهیم. در نتیجه، با اندکی تلاش می‌توان تعریفی بازگشتی جهت محاسبه حاصلضرب به دست آورد که حل مسئله را به صورت تقسیم و غلبه ممکن کند. شبه‌کد آن به صورت زیر است.

```
Alg. Zarb_Chandjomleye_ToQ(A, B, n)
If len(A) == 1
   return C[0] = A[0] * B[0]
C[0:2n-2] = 0
k = n/2;
A_c = A[0: k/2 -1]
A_r = A[k/2: n -1]
B_c = B[0: k/2 -1]
B_r = B[k/2: n -1]
C_c = Zarb_Chandjomleye_ToQ(A_c, B_c, k)
C_rc = Zarb_Chandjomleye_ToQ(A_r, B_c, k)
C_cr = Zarb_Chandjomleye_ToQ(A_c, B_r, k)
C_r = Zarb_Chandjomleye_ToQ(A_r, B_r, k)
C[0 : n - 2] += C_c
C[k : k + n - 2] += C_rc + C_cr
C[n : 2n - 2] += C_r
Return C
```

همان‌طور که از کد بالا پیداست $x^{\frac{n}{2}}$ یا x^n معادل شیفت مقادیر متناظر به اندازه k یا n به سمت راست در آرایهٔ حاصل است. کد سی روش تقسیم و غلبه حل ضرب دو چندجمله‌ای به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>
// Tabe Jame do Chandjomelyeha
void jameChandjomelyeha(int *A, int *B, int *hasel, int n) {
   for (int i = 0; i < n; i++) {
      hasel[i] = A[i] + B[i];
   }
}


// Tabe tfriqe do Chandjomelyeha
void tafriqChandjomelyeha(int *A, int *B, int *hasel, int n) {
   for (int i = 0; i < n; i++) {
      hasel[i] = A[i] - B[i];
   }
}


// Tabe zarbe do Chandjomelyeha
void zarbChanjomlyeToQ(int *A, int *B, int *hasel, int n) {
   if (n == 1) {
      hasel[0] = A[0] * B[0];
      return;
   }
```

```
    int k = n / 2;


    // Nime chap o rast A o B
    int *A_c = A;
    int *A_r = A + k;
    int *B_c = B;
    int *B_r = B + k;


    // araye mvqt jahate mqadire miani
    int *hasel_c = (int *)calloc(n, sizeof(int));   // A_c * B_c
    int *hasel_r = (int *)calloc(n, sizeof(int));   // A_r * B_r
    int *A_cB_r = (int *)calloc(k, sizeof(int));    // A_c * B_r
    int *A_rB_c = (int *)calloc(k, sizeof(int));    // A_r * B_c
    zarbChanjomlyeToQ(A_c, B_c, hasel_c, k); // A_c * B_c
    zarbChanjomlyeToQ(A_r, B_r, hasel_r, k);  // A_r * B_r
    zarbChanjomlyeToQ(A_c, B_r, A_cB_r, k);
    zarbChanjomlyeToQ(A_r, B_c, A_rB_c, k);


    // Tarkib natayeje miani
    for (int i = 0; i < n; i++) {
        hasel[i] = 0;
    }


    // bakhshe chap
    for (int i = 0; i < n; i++) {
        hasel[i] += hasel_c[i];
    }


    //
    for (int i = 0; i < n; i++) {
        hasel[i + n] += hasel_r[i];
    }


    //
    for (int i = 0; i < 2 * k; i++) {
        hasel[i + k] += hasel_c[i] + hasel_r[i];
    }


    // Rahasazi Hafeze
    free(hasel_c);
    free(hasel_r);
    free(A_cB_r);
    free(A_rB_c);
}
```

```
int main() {
   int n = 4; // darjeye chandjomeleye
   int A[] = {1, 1, 1, 1}; // chandjomeleye A
   int B[] = {1, 1, 1, 1}; // chandjomeleye B


   int *hasel = (int *)calloc(2 * n, sizeof(int));


   // zarbe do chandjomleye
   zarbChanjomlyeToQ(A, B, hasel, n);
   // nemayeshe natayej
   printf("Hasel zarb do chandjomeleye: ");
   for (int i = 0; i < 2 * n - 1; i++) {
      printf("%d ", hasel[i]);
   }
   printf("\n");
   // Rahasazi Hafeze
   free(hasel);
   return 0;
}
```

جهت ضرب به روش تقسیم و غلبهٔ معرفی شدهٔ دو چند جمله‌ای به طول n تابع بازگشتی،چهار ضرب دو چندجمله‌ای با اندازه کوچکتر محاسبه و نتایج با مرتبه‌ای از n ترکیب می‌شوند. در نتیجیجه تابع بازگشتی زمان اجرا به صورت زیر است.

$$z(n) = ۴z\left(\frac{n}{۲}\right) + cn$$

با استفاده از قضیه اصلی از مرتبهٔ $\theta\left(n^{۲}\right)$ است. . پس در مرتبهٔ زمانی بهبودی مشاهده نمی‌شود.

تقسیم و حل- روش بهینه‌تر

اشاره شد که تقسیم چندجمله‌ای به دو نیمه در زمان اجرا تاثیر آن‌چنانی نداشت. اما تقسیم مذکور راه را برای کاهش زمان ضرب چندجمله‌ای گشود. بار دیگر ضرب را بررسی می‌کنیم.

$$r(x) = p(x)q(x) = \left(p_c(x) + x^{\frac{n}{۲}} p_r(x)\right)\left(q_c(x) + x^{\frac{n}{۲}} q_r(x)\right)$$

$$= p_c(x)q_c(x) + x^{\frac{n}{۲}}[p_r(x)q_c(x) + p_c(x)q_r(x)] + x^{n} p_r(x)\, q_r(x)$$

حال تعاریف چندجمله‌ای جدیدی را در نظر بگیریم.

- $r_c(x) = p_c(x)q_c(x)$

- $r_r(x) = p_r(x)\, q_r(x)$
- $r_m(x) = \big(p_c(x) + p_r(x)\big)\big(q_c(x) + q_r(x)\big)$

دو مورد متقدم در تقسیم و غلبه قبلی نیز وجود داشتند. اما مورد اخیر متفاوت است و در واقع موجب کاهش تعداد ضرب‌هاست. آن را بیشتر تحلیل می‌کنیم.

$$r_m(x) = \big(p_c(x) + p_r(x)\big)\big(q_c(x) + q_r(x)\big)$$

$$= p_c(x)q_c(x) + [p_r(x)q_c(x) + p_c(x)q_r(x)] + p_r(x)\, q_r(x)$$

با جانشینی متغیرهای تعریف شده داریم:

$$= r_c(x) + [p_r(x)q_c(x) + p_c(x)q_r(x)] + \ r_r(x)$$

پس

$$r_m(x) = r_c(x) + [p_r(x)q_c(x) + p_c(x)q_r(x)] + \ r_r(x)$$

با جابجایی داریم:

$$[p_r(x)q_c(x) + p_c(x)q_r(x)] = r_m(x) - r_c(x) - \ r_r(x)$$

سخن کوتاه با محاسبهٔ $r_c(x)$ و $r_r(x)$ و $r_m(x)$ می‌توان به جای چهار عملیات ضرب دو چندجمله‌ای از سه عملیات ضرب دو زیرچندجمله‌ای بهره برد. شبه‌کد آن به صورت زیر است.

```
Alg. Zarb_Chandjomleye_ToQ(A, B, n)
If len(A) == 1
   return C[0] = A[0] * B[0]
C[0:2n-2] = 0
k = n/2;
A_c = A[0: k/2 -1]
A_r = A[k/2: n -1]
B_c = B[0: k/2 -1]
B_r = B[k/2: n -1]
C_c = Zarb_Chandjomleye_ToQ(A_c, B_c, k)
C_m = Zarb_Chandjomleye_ToQ(A_c + B_c, A_r + B_r, k)
C_r = Zarb_Chandjomleye_ToQ(A_r, B_r, k)
C[0 : n - 2] += C_c
C[k : k + n - 2] += C_m - C_c - C_r
C[n : 2n - 2] += C_r
Return C
```

الگوریتم ضرب چندجمله‌ای تقسیم و غلبهٔ معرفی شده در زبان سی به صورت زیر پیاده می‌شود.

```
#include <stdio.h>
#include <stdlib.h>
```

```
// Tabe Jame do Chandjomelye
void jameChandjomelyeha(int *A, int *B, int *hasel, int n) {
   for (int i = 0; i < n; i++) {
      hasel[i] = A[i] + B[i];
   }
}


// Tabe tfriqe do Chandjomelye
void tafriqChandjomelyeha(int *A, int *B, int *hasel, int n) {
   for (int i = 0; i < n; i++) {
      hasel[i] = A[i] - B[i];
   }
}


// Tabe zarbe do Chandjomelye
void zarbChanjomlyeToQ(int *A, int *B, int *hasel, int n) {
   if (n == 1) {
      hasel[0] = A[0] * B[0];
      return;
   }


   int k = n / 2;


   int *A0 = A;
   int *A1 = A + k;
   int *B0 = B;
   int *B1 = B + k;


   int *A0B0 = (int *)calloc(2 * k, sizeof(int));
   int *A1B1 = (int *)calloc(2 * k, sizeof(int));
   int *A0A1 = (int *)calloc(k, sizeof(int));
   int *B0B1 = (int *)calloc(k, sizeof(int));
   int *A0A1B0B1 = (int *)calloc(2 * k, sizeof(int));


   zarbChanjomlyeToQ(A0, B0, A0B0, k);
   zarbChanjomlyeToQ(A1, B1, A1B1, k);


   jameChandjomelyeha(A0, A1, A0A1, k);
   jameChandjomelyeha(B0, B1, B0B1, k);


   zarbChanjomlyeToQ(A0A1, B0B1, A0A1B0B1, k);


   for (int i = 0; i < n; i++) {
      hasel[i] = 0;
   }
```

```
    for (int i = 0; i < 2 * k; i++) {
        hasel[i] += A0B0[i];
        hasel[i + n] += A1B1[i];
        hasel[i + k] += A0A1B0B1[i] - A0B0[i] - A1B1[i];
    }
    free(A0B0);
    free(A1B1);
    free(A0A1);
    free(B0B1);
    free(A0A1B0B1);
}


int main() {
    int n = 4; // darajeye Chandjomelye-ha
    int A[] = {1, 1, 1, 1}; // Chandjomelye A
    int B[] = {1, 1, 1, 1}; // Chandjomelye B


    int *hasel = (int *)calloc(2 * n, sizeof(int));


    zarbChanjomlyeToQ(A, B, hasel, n);


    printf("Chanjomleye hasel: ");
    for (int i = 0; i < 2 * n - 1; i++) {
        printf("%d ", hasel[i]);
    }
    printf("\n");


    free(hasel);
    return 0;
}
```

زمان اجرای الگوریتم برابر با مقدار زیر است.

$$z(n) = ۳z\left(\frac{n}{۲}\right) + cn = \theta\left(n^{\log ۳}\right) = \theta\left(n^{۱.۵۸}\right)$$

که بهبود نسبتا خوبی به ضرب معمولی ضرب است.

تمرین- ضرب دو عدد بزرگ با تعداد ارقام زیاد را چگونه انجام می‌دهید؟ کد آن را پیاده کنید.

تمرین- بررسی شود که الگوریتم مناسبتری وجود دارد؟ روش توضیح داده شود.

تبدیل فوریه سریع

بسیاری از مسائل رایانشی از روش‌های تبدیل فوریه یا روش‌های طیفی استفاده می‌برند. در چنین مسائلی تبدیل فوریه ابزار محاسباتی کارآمدی جهت کار با داده است. یا با تبدیل فوریه و انتقال به حوزهٔ آنالیز، داده‌ها در حوزه مذکور محل تحلیل و استفاده‌اند. در واقع، سیگنال در حوزه زمان را می‌توان به سیگنال در حوزه بسامد (فرکانس) تبدیل کرد. نگاشت از حوزه زمان به بسامد و بالعکس با تبدیل فوریه ممکن می‌شود. در صورتی که تابع در حوزه زمان را با h(t) و در حوزه بسامد با H(f) نمایش داریم. نحوهٔ محاسبه به صورت زیر است.

$$\mathrm{H(f)} = \int_{-\infty}^{+\infty} h(t) e^{۲\pi i f t} dt$$

$$h(t) = \int_{-\infty}^{+\infty} H(f) e^{-۲\pi i f t} dt$$

به طوری که t ثانیه، بسامد f دور در ثانیه یا هرتز (واحد بسامد) است. اگر h متر باشد، آن‌گاه H بر حسب معکوس طول موج (دور در متر است). فیزیکداناو ریاضیدانان از بشسامد زاویه‌ای ω استفاده می‌کنند که واحد آن رادیان و برابر با $\omega = ۲\pi f$ است.

همان‌طور که از معادله پیداست، عملگر تبدیل فوریه عملگری خطی است (به چه معنا؟).

اما، در رایانه مقادیر پیوسته نیست. در نتیجه برداری گسسته و n مقداری معرف f است که به حوزه بسامد با مقادیر گسسته نگاشت و با F نمایش داده می‌شود. با فرض $f = (f_0, f_۱, \cdots, f_{n-۱})$ و $\omega_n = e^{\frac{۲\pi i}{n}}$ تبدیل فوریه گسسته و معکوس آن به صورت زیر هستند.

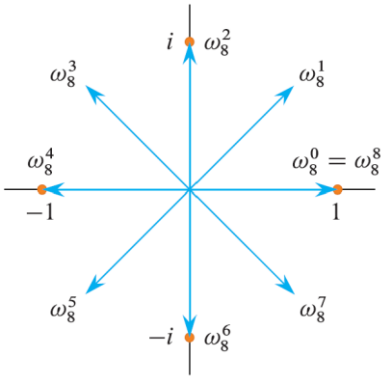
$$F_k = F(\omega_n^k) = \sum_{j=0}^{n-۱} f_j \omega_n^{kj} = \sum_{j=0}^{n-۱} f_j e^{\frac{۲\pi i k j}{n}}$$

به طوری که برای تک تک مقادیر $F = (F_0, F_۱, \cdots, F_{n-۱})$ از معادله بالا استفاده می‌شود. هم‌چنین، معکوس آن عبارت است از

$$f_k = \frac{۱}{n} \sum_{j=0}^{n-۱} F_j e^{\frac{-۲\pi i k j}{n}}$$

مسئله را می‌توان از منظری دیگر نگریست. استفاده و محاسبهٔ ریشهٔ n-ام یک، یا $\omega^n = ۱$ دارای کاربردهای فراوان در فیزیک، مهندسی برق، مهندسی رایانه، و پردازش سیگنال است. عدد یک دارای دقیقا n ریشهٔ مختلط $e^{۲\pi ik/n}, k = 0,۱,\cdots,n-۱$ است. از طرف دیگر، می‌توان اعداد نمایی مختلط را از ترکیبی از مقادیر مثلثاتی نوشت. چنین نمایشی در قامت معادله اوئلر معرفی می‌شود.

$$e^{i\theta} = \cos\theta + i\sin\theta$$



ویژگی‌ها

لم خنثی‌سازی: به ازای اعداد صحیح $n > 0$، $k \geq 0$، و $d>0$ داریم $\omega_{dn}^{dk} = \omega_n^k$.

اثبات-

$$\omega_{dn}^{dk} = \left(e^{۲\pi i/dn}\right)^{dk} = \left(e^{۲\pi i/n}\right)^k = \omega_n^k.$$

نتایج-

أ. طبق لم خنثی‌سازی $(\omega_n^k)^۲ = \omega_{n/۲}^k$.

ب. همچنین $\left(\omega_n^{k+n/۲}\right)^۲ = \omega_n^{۲k+n} = \omega_n^{۲k}\,\omega_n^n = \omega_n^{۲k} = (\omega_n^k)^۲ = \omega_{n/۲}^k$.

تمرین- نشان دهید $\sum_{j=0}^{n-1}(\omega_n^k)^j = 0$.

با یادآوری نمایش چندجمله‌ای $A(x) = \sum_{j=0}^{n-1} a_j x^j$ می‌توان تبدیل فوریه گسسته را به صورت چندجمله‌ای در نظر آورد. جهت یکسان‌سازی نمایش با بحث سیگنال تابع را به صورت $F(x) = \sum_{j=0}^{n-1} f_j x^j$ باز می‌نویسیم. توجه کنید f کوچک در حوزهٔ زمان و F بزرگ در حوزهٔ بسامد است.

به دنبال ارزیابی تابع در n ریشه مختلط n-ام واحد. با فرض $f = (f_0, f_1, \cdots, f_{n-1})$ و $\omega_n = e^{\frac{2\pi i}{n}}$

$$F_k = F(\omega_n^k) = \sum_{j=0}^{n-1} f_j \omega_n^{kj} = \sum_{j=0}^{n-1} f_j e^{\frac{2\pi i k j}{n}}$$

$$F = (F_0, F_1, \cdots, F_{n-1})$$

معادل تبدیل فوریه گسسته بردار ضرائب f است. می‌توان تابع را به صورت $F = DFT_n(f)$ نوشت. شبه‌کد تبدیل فوریه گسسته در ادامه آمده است.

```
DFT(f[], n)
  F[0: n – 1] = 0
  W [0:n-1] = 1
  ω_n = e^(2πi/n)
  for k = 1 : n – 1
    W[k] = W[k − 1] ∗ ω_n

  for k = 0 : n - 1
    for j = 0 : n - 1
      F[k] = F[k] + f[j] ∗ W[k] ∗ W[j]

  return F
```

کد آن به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>


void DFT(double complex *x, double complex *F, int N) {
  for (int k = 0; k < N; k++) {
    F[k] = 0; // mqdardehi sefre
    for (int n = 0; n < N; n++) {
      double theta = -2.0 * M_PI * k * n / N; // Mohasebeye zaveye (theta)
      F[k] += x[n] * cexp(I * theta);      // Jame tajmiei
    }
  }
```

```
}


int main() {
   int n = 128;              // andazeye vorodi tavani az do
   double MizanNemonebaradri = 128.0; // sor'at nemonebardari  ya samplinggRate (Hz)
   double bozorghi = 1.0;     // andazeye mowje sinosi
   double basamad = 10.0;     // basamad mowje sinosi (Hz)
/*
     double basamad2 = 50.0;     // basamad mowje sinosi (Hz)
   double bozorghi2 = 3.0;     // andazeye mowje sinosi
     double basamad3 = 27.0;     // basamad mowje sinosi (Hz)
   double bozorghi3 = 4.0;     // andazeye mowje sinosi
*/
   // Tolid signal (mowje) sinosi
   complex double *a = (complex double *)malloc(n * sizeof(complex double));
   for (int i = 0; i < n; i++) {
      double t = i / MizanNemonebaradri; // Ghame zaman
                                 a[i] = bozorghi * sin(2 * M_PI * basamad * t) + 0.0 * I; // sinos mqdar
haghighi
      // a[i] = bozorghi * sin(2 * M_PI * basamad * t) + bozorghi2 * sin(2 * M_PI * basamad2 * t) +
0.0 * I; // sinos mqdar haghighi
   }
                /*
                for (int i = 0; i < n; i++) {
      double t = i / MizanNemonebaradri; // Ghame zaman
      a[i] += bozorghi3 * sin(2 * M_PI * basamad3 * t)  + 0.0 * I; // sinos mqdar haghighi
   }
                */


   // Zakhireye signal zaman dar file
   FILE *FileZaman = fopen("hoze_zaman.dat", "w");
   for (int i = 0; i < n; i++) {
      double t = i / MizanNemonebaradri; // Ghame zaman
      fprintf(FileZaman, "%lf %lf\n", t, creal(a[i]));
   }
   fclose(FileZaman);


   // farakhani FFT
     double complex F[n];
   DFT(a, F, n);


   // Zakhireye signal basamad dar file
   FILE *FileBasamad = fopen("hoze_basamad.dat", "w");
   for (int i = 0; i < n / 2; i++) {
      double freq = i * MizanNemonebaradri / n; // basamadhaye mosbat
      fprintf(FileBasamad, "%lf %lf\n", freq, cabs(F[i])); // Bozorghi hasele FFT
   }
   fclose(FileBasamad);
```

```
    // rahasazi hafeze
    free(a);

    return 0;
}
```

مرتبهٔ آن از $O\left(n^{۲}\right)$ است. کولی و تاکی حدود ۱۳۴۰ شمسی تبدیل فوریه سریع را معرفی کردند که سرعت محاسبه را افزایش داد. البته، بعدا روشن افراد دیگری نیز تبدیل فوریه سریع را پیش‌تر گسترش داده بودند. در ادامه بر اساس تحلیل شبیه تحلیل دانیلسن در سال ۱۳۲۰ شمسی روش حل سریع را اثبات می‌کنیم. تبدیل فوریه سریع استفاده از استراتژی تقسیم و غلبه با استفاده جداگانه از اندیس‌های زوج و فرد ضرائب $F(x)$ جهت تعریف دو چند جمله‌ای زوج و مرتب استفاده می‌شود. به عبارت دیگر،

$$F_k = F(\omega_n^k) = \sum_{j=0}^{n-۱} f_j \omega_n^{kj} = \sum_{j=0}^{n-۱} f_j e^{\frac{۲\pi i k j}{n}}$$

$$= \sum_{j=0}^{\frac{n}{۲}-۱} f_{۲j} e^{\frac{۲\pi i k (۲j)}{n}} + \sum_{j=0}^{\frac{n}{۲}-۱} f_{۲j+۱} e^{\frac{۲\pi i k (۲j+۱)}{n}}$$

$$= \sum_{j=0}^{n/۲-۱} f_{۲j} e^{\frac{۲\pi i k j}{n/۲}} + \omega^k \sum_{j=0}^{n/۲-۱} f_{۲j+۱} e^{\frac{۲\pi i k j}{n/۲}}$$

$$= F_k^{[ز]} + \omega^k F_k^{[ف]}$$

$F^{[ف]}(x)$ و $F^{[ز]}(x)$

$$F^{[ز]}(x) = f_0 + f_۲ x + f_۴ x^۲ + \cdots + f_{n-۲} x^{\frac{n}{۲}-۱}$$

$$F^{[ف]}(x) = f_۱ + f_۳ x + f_۵ x^۲ + \cdots + f_{n-۱} x^{\frac{n}{۲}-۱}$$

به طوری که

$$F(x) = F^{[ز]}\left(x^۲\right) + x F^{[ف]}(x^۲)$$

شبه‌کد الگوریتم تبدیل فوریه سریع به صورت زیر است.

$Alg\ FFT - bazgashti(f[\ \ ], n)$
$if\ n\ ==1$
$\quad return\ f$
$\omega_n = e^{\frac{2\pi i}{n}}$
$\omega = 1$
$f^{[ز]} = \left(f_0, f_۲, f_۴, \cdots, f_{n-۲}\right)$
$f^{[ف]} = (f_۱, f_۳, f_۵, \cdots, f_{n-۱})$
$F^{[ز]} = FFT - bazgashti\left(f^{[ز]}\right)$
$F^{[ف]} = FFT - bazgashti(f^{[ف]})$
$for\ k = 0 : n/۲ - ۱$
$mvqt = \omega F_k^{[ف]}$
$F_k = F_k^{[ز]} + mvqt$
$F_{k+(n/۲)} = F_k^{[ز]} - mvqt$
$\omega = \omega * \omega_n$
$return\ F$

دو تابع بازگشتی بر ضرائب بخش‌های زوج و فرد اعمال می‌شوند. چون $k = 0, \ldots, \frac{n}{2} - ۱$

$$\left.\begin{aligned} F_k^{[ز]} &= F^{[ز]}\left(\omega_{\frac{n}{۲}}^{k}\right) \\ F_k^{[ف]} &= F^{[ف]}\left(\omega_{\frac{n}{۲}}^{k}\right) \end{aligned}\right\} \xrightarrow{\omega_{\frac{n}{۲}}^{k}=\omega_n^{۲k}\ \text{دلیل به (چرا؟)}} \left\{\begin{aligned} F_k^{[ز]} &= F^{[ز]}\left(\omega_n^{۲k}\right) \\ F_k^{[ف]} &= F^{[ف]}\left(\omega_n^{۲k}\right) \end{aligned}\right.$$

همچنین، برای F_0، ...، $F_{\frac{n}{۲}-۱}$

$$F_k = F_k^{[ز]} + \omega_n^k F_k^{[ف]} = F^{[ز]}\left(\omega_n^{۲k}\right) + \omega_n^k F^{[ف]}\left(\omega_n^{۲k}\right) = \mathrm{F}(\omega_n^k)$$

و برای $F_{\frac{n}{۲}}$ تا $F_{n-۱}$ و k=0,...,$\frac{\mathrm{n}}{۲} - ۱$

$$F_{k+(n/2)} = F_k^{[زوج]}\left(\omega_n^{2k}\right) - \omega_n^k F_k^{[فرد]}\left(\omega_n^{2k}\right) \xrightarrow{-\omega_n^k = \omega_n^{k+\frac{n}{2}}}$$

$$= F_k^{[زوج]}\left(\omega_n^{2k}\right) + \omega_n^{k+\frac{n}{2}} F_k^{[فرد]}\left(\omega_n^{2k}\right) \xrightarrow{\omega_n^{2k} = \omega_n^{2k+n}}$$

$$= F^{[زوج]}\left(\omega_n^{2k+n}\right) + \omega_n^{k+\frac{n}{2}} F^{[فرد]}\left(\omega_n^{2k+n}\right) \xrightarrow{F(x)=F^{[زوج]}\left(x^2\right)+xF^{[فرد]}(x^2)}$$

$$= \mathrm{F}\left(\omega_n^{k+n/2}\right)$$

برنامه سی تبدیل فوریه سریع به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>


// Tabe FFT bazgashti
void FFT_bazgashti(complex double *a, int n) {
   if (n == 1) {
      return; // Paye
   }


   int nim = n / 2;


   // Tqsim
   complex double *zowj = (complex double *)malloc(nim * sizeof(complex double));
   complex double *fard = (complex double *)malloc(nim * sizeof(complex double));


   for (int i = 0; i < nim; i++) {
      zowj[i] = a[2 * i];
      fard[i] = a[2 * i + 1];
   }


   // Qalabe
   FFT_bazgashti(zowj, nim);
   FFT_bazgashti(fard, nim);


   // Trkib
   for (int k = 0; k < nim; k++) {
      complex double t = cexp(-2.0 * I * M_PI * k / n) * fard[k]; // Twiddle factor
      a[k] = zowj[k] + t;
      a[k + nim] = zowj[k] - t;
   }
```

```
    // Rahasazi Hafezeye takhsisi
    free(zowj);
    free(fard);
}
/*
int main() {
    int n = 8; // andazeye vorodi tavani az do


    // vorodi be sorat adad mokhtalet
    complex double a[] = {1.0, 2.0, 3.0, 4.0, 0.0, 0.0, 0.0, 0.0};


    printf("arraye vorodi:\n");
    for (int i = 0; i < n; i++) {
        printf("%.2f + %.2fi\n", creal(a[i]), cimag(a[i]));
    }


    // Farakhani FFT_bazgashti
    FFT_bazgashti(a, n);


    printf("\nhasele FFT_bazgashti:\n");
    for (int i = 0; i < n; i++) {
        printf("%.2f + %.2fi\n", creal(a[i]), cimag(a[i]));
    }


    return 0;
}
*/
/*
int main()
{
    int n = 16; // andazeye vorodi tavani az do
    double MizanNemonebaradri = 16.0; // sor'at nemonebardari  ya samplinggRate (Hz)
    double basamad = 2.0;    // basamad mowje sinosi (Hz)
    double bozorghi = 1.0;    // andazeye mowje sinosi


    // Tolid signal (mowje) sinosi
    complex double *a = (complex double *)malloc(n * sizeof(complex double));
    for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman
        a[i] = bozorghi * sin(2 * M_PI * basamad * t) + 0.0 * I; // sinos mqdar haghighi
    }


    printf("mowje sinosi vorodi:\n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %.2f + %.2fi\n", i, creal(a[i]), cimag(a[i]));
```

```
    }

    // farakhani FFT
    FFT_bazgashti(a, n);

    // Chape hasel FFT
    printf("\nhasele FFT:\n");
    for (int i = 0; i < n; i++) {
        printf("F[%d] = %.2f + %.2fi\n", i, creal(a[i]), cimag(a[i]));
    }

    // rahasazi hafeze
    free(a);

    return 0;
}
*/
/*
int main() {
    int n = 128;            // andazeye vorodi tavani az do
    double MizanNemonebaradri = 128.0; // sorat nemonebardari  ya samplinggRate (Hz)
    double basamad = 10.0;    // basamad mowje sinosi (Hz)
    double bozorghi = 1.0;    // amplitude ya andazeye mowje sinosi

    // Tolid signal (mowje) sinosi
    complex double *a = (complex double *)malloc(n * sizeof(complex double));
    for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman (Time step)
        a[i] = bozorghi * sin(2 * M_PI * basamad * t) + 0.0 * I; // sinos mqdar haghighi
    }

    // Zakhireye signal zaman dar file
    FILE *FileZaman = fopen("hoze_zaman.dat", "w");
    for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman
        fprintf(FileZaman, "%lf %lf\n", t, creal(a[i]));
    }
    fclose(FileZaman);

    // farakhani FFT
    FFT_bazgashti(a, n);

    // Zakhireye signal basamad dar file
    FILE *FileBasamad = fopen("hoze_basamad.dat", "w");
    for (int i = 0; i < n; i++) {
        double freq = i * MizanNemonebaradri / n; // game basamad
```

```
    fprintf(FileBasamad, "%lf %lf\n", freq, cabs(a[i])); // Bozorghi hasele FFT
  }
  fclose(FileBasamad);


  // Rasm ba Gnuplot
  FILE *gnuplot = popen("gnuplot -persistent", "w");
  fprintf(gnuplot, "set multiplot layout 1,2 title 'Zaman and basamad Domain'\n");
  fprintf(gnuplot, "set xlabel 'Zaman (s)'\n");
  fprintf(gnuplot, "set ylabel 'bozorghi'\n");
  fprintf(gnuplot, "plot 'hoze_zaman.dat' with lines title 'Hoze Zaman'\n");
  fprintf(gnuplot, "set xlabel 'basamad (Hz)'\n");
  fprintf(gnuplot, "set ylabel 'bozorghi'\n");
  fprintf(gnuplot, "plot 'hoze_basamad.dat' with lines title 'Hoze basamad'\n");
  fprintf(gnuplot, "unset multiplot\n");
  pclose(gnuplot);


  // rahasazi hafeze
  free(a);


  return 0;
}
*/
/*
int main() {
  int n = 128;          // andazeye vorodi tavani az do
  double MizanNemonebaradri = 128.0; // sor'at nemonebardari  ya samplinggRate (Hz)
  double basamad = 10.0;   // basamad mowje sinosi (Hz)
  double bozorghi = 1.0;    // amplitude ya andazeye mowje sinosi


  // Tolid signal (mowje) sinosi
  complex double *a = (complex double *)malloc(n * sizeof(complex double));
  for (int i = 0; i < n; i++) {
    double t = i / MizanNemonebaradri; // Ghame zaman
    a[i] = bozorghi * sin(2 * M_PI * basamad * t) + 0.0 * I; // sinos mqdar haghighi
  }


  // Zakhireye signal zaman dar file
  FILE *FileZaman = fopen("hoze_zaman.dat", "w");
  for (int i = 0; i < n; i++) {
    double t = i / MizanNemonebaradri; // Ghame zaman
    fprintf(FileZaman, "%lf %lf\n", t, creal(a[i]));
  }
  fclose(FileZaman);


  // farakhani FFT
  FFT_bazgashti(a, n);
```

```
    // Zakhireye signal basamad dar file
    FILE *FileBasamad = fopen("hoze_basamad.dat", "w");
                for (int i = 0; i <= n / 2; i++) {
                            double freq = i * MizanNemonebaradri / n; // basamadhaye mosbat
                            fprintf(FileBasamad, "%lf %lf\n", freq, cabs(a[i])); // Bozorghi hasele FFT
                }
    fclose(FileBasamad);


    // Rasm ba Gnuplot
    FILE *gnuplot = popen("gnuplot -persistent", "w");
    fprintf(gnuplot, "set multiplot layout 1,2 title 'Hoze zaman o basamad'\n");
    fprintf(gnuplot, "set xlabel 'Zaman (s)'\n");
    fprintf(gnuplot, "set ylabel 'Damane'\n");
    fprintf(gnuplot, "plot 'hoze_zaman.dat' with lines title 'Hoze Zaman'\n");
    fprintf(gnuplot, "set xlabel 'basamad (Hz)'\n");
    fprintf(gnuplot, "set ylabel 'Bozorghi'\n");
    fprintf(gnuplot, "plot 'hoze_basamad.dat' with lines title 'Hoze basamad'\n");
    fprintf(gnuplot, "unset multiplot\n");
    pclose(gnuplot);


    // rahasazi hafeze
    free(a);
    return 0;
}
*/
int main() {
    int n = 128;            // andazeye vorodi tavani az do
    double MizanNemonebaradri = 128.0; // sor'at nemonebardari  ya sampling Rate (Hz)
    double bozorghi = 1.0;     // andazeye mowje sinosi
    double basamad = 10.0;     // basamad mowje sinosi (Hz)
                double basamad2 = 50.0;    // basamad mowje sinosi (Hz)
    double bozorghi2 = 3.0;     // andazeye mowje sinosi
                double basamad3 = 27.0;    // basamad mowje sinosi (Hz)
    double bozorghi3 = 4.0;     // andazeye mowje sinosi


    // Tolid signal (mowje) sinosi
    complex double *a = (complex double *)malloc(n * sizeof(complex double));
    for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman
        a[i] = bozorghi * sin(2 * M_PI * basamad * t) + bozorghi2 * sin(2 * M_PI * basamad2 * t) + 0.0
* I; // sinos mqdar haghighi
    }
                for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman
        a[i] += bozorghi3 * sin(2 * M_PI * basamad3 * t)  + 0.0 * I; // sinos mqdar haghighi
    }


    // Zakhireye signal zaman dar file
    FILE *FileZaman = fopen("hoze_zaman.dat", "w");
```

```
    for (int i = 0; i < n; i++) {
        double t = i / MizanNemonebaradri; // Ghame zaman
        fprintf(FileZaman, "%lf %lf\n", t, creal(a[i]));
    }
    fclose(FileZaman);


    // farakhani FFT
    FFT_bazgashti(a, n);


    // Zakhireye signal basamad dar file
    FILE *FileBasamad = fopen("hoze_basamad.dat", "w");
    for (int i = 0; i < n / 2; i++) {
        double freq = i * MizanNemonebaradri / n; // basamadhaye mosbat
        fprintf(FileBasamad, "%lf %lf\n", freq, cabs(a[i])); // Bozorghi hasele FFT
    }
    fclose(FileBasamad);


    // rahasazi hafeze
    free(a);


    return 0;
}
```

تابع زمان اجاری بازگشتی الگوریتم تبدیل فوریه سریعع به صورت زیر خواهد بود. درستی گزاره را بررسی و مقدار را به صورت تابعی از n بدست آورید.

$$Z(n) = ۲Z\left(\frac{n}{۲}\right) + \theta(n)$$

ضرب دو ماتریس

ضرب ماتریس‌ها معمولا عملیاتی پر کاربرد در محاسبات است. اگر دو ماتریس مربع و هم‌اندازه باشند ضرب آنها ممکن است. فرض می‌کنیم دو ماتریس A و B با اندازهٔ $n \times n$ را داریم. ضرب آنها، ماتریسی با نام C است که مدحل (i,j) آن برابر با ضرب داخلی ردیف i–ام ماتریس A در ستون j–ام ماتریس B است، یا

$$C[i,j] = \sum_{k=0}^{n-۱} A[i,k] \times B[k,j]$$

الگوریتم معمولی و ساده ضرب دو ماتریس به صورت زیر است.

```
Alg. ZarbMatrix(A,B)
   C = [0]n× n
   for i = 0:n-1
      for j = 0:n-1
         for k = 0:n-1
            C[i,j] = C[i,j] + A[i,k]*B[k,j]
```

کد آن در سی به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>


// tabe takhsis hafeze be matrix
int** takhsisMatrix(int t_radifha, int t_soton) {
   int** matrix = (int**)malloc(t_radifha * sizeof(int*));
   for (int i = 0; i < t_radifha; i++) {
      matrix[i] = (int*)malloc(t_soton * sizeof(int));
   }
   return matrix;
}
// Tabe rahasazi hafezeye eshqali matrix
void rahasaziMatrix(int** matrix, int t_radifha) {
   for (int i = 0; i < t_radifha; i++) {
      free(matrix[i]);
   }
   free(matrix);
}


// Tabe zarb do matrix
int** zarbmatrixha_sade(int** A, int** B, int t_radifhaA, int t_sotonA, int t_sotonB) {
   int** C = takhsisMatrix(t_radifhaA, t_sotonB);
   for (int i = 0; i < t_radifhaA; i++) {
      for (int j = 0; j < t_sotonB; j++) {
         C[i][j] = 0;
         for (int k = 0; k < t_sotonA; k++) {
                                                C[i][j] += A[i][k] * B[k][j];
                                          }


                                          printf("%d\t", C[i][j]);
                              }
                              printf("\n");
                  }
}


int main() {
   int n = 16;
                  int** A = takhsisMatrix(n, n);
                  for(int i = 0; i < n; i++){
                              for (int j = 0; j < n; j++){
```

```
                                        A[i][j] = (i==j)?1:0;
                                }
                        }


                        int** B = takhsisMatrix(n, n);
                        for(int i = 0; i < n; i++){
                                for (int j = 0; j < n; j++){
                                        B[i][j] = j;
                                }
                        }
                        int** C = zarbmatrixha_sade(A, B, n, n, n);
                        rahasaziMatrix(A, n);
                        rahasaziMatrix(B, n);
                        rahasaziMatrix(C, n);
        return 0;
}
```

کد الگوریتم ساده دارای مرتبه زمان $O\left(n^{۳}\right)$ است. جهت حل مسئلهٔ مذکور نیز می‌توان از تقسیم و غلبه بهره برد. در ادامه روش تقسیم و غلبه را بر مبنای تقسیم و غلبه گزارش می‌کنیم.

ضرب ماتریسی با روش تقسیم و غلبه- تلاش نخست

دو ماتریس ورودی و همچنین ماتریس حاصل را به چهار زیرماتریس تقسیم می‌کنیم. به سخن دیگر،

$$\mathrm{A}=\begin{bmatrix}A_{۱۱} & A_{۱۲}\\ A_{۲۱} & A_{۲۲}\end{bmatrix}\text{ و }\mathrm{B}=\begin{bmatrix}B_{۱۱} & B_{۱۲}\\ B_{۲۱} & B_{۲۲}\end{bmatrix}\text{ و }\mathrm{C}=\begin{bmatrix}C_{۱۱} & C_{۱۲}\\ C_{۲۱} & C_{۲۲}\end{bmatrix}$$

هر کدام از زیرماتریس‌های حاصل دارایا ابعاد $\frac{n}{۲}\times\frac{n}{۲}$ هستند. حال جهت محاسبه می‌توان به صورت زیر عمل کرد.

$$\begin{bmatrix}C_{۱۱} & C_{۱۲}\\ C_{۲۱} & C_{۲۲}\end{bmatrix}=\begin{bmatrix}A_{۱۱} & A_{۱۲}\\ A_{۲۱} & A_{۲۲}\end{bmatrix}\begin{bmatrix}B_{۱۱} & B_{۱۲}\\ B_{۲۱} & B_{۲۲}\end{bmatrix},$$

به طوری‌که،

$$C_{۱۱}=A_{۱۱}B_{۱۱}+A_{۱۲}B_{۲۱},$$
$$C_{۱۲}=A_{۱۱}B_{۱۲}+A_{۱۲}B_{۲۲},$$
$$C_{۲۱}=A_{۲۱}B_{۱۱}+A_{۲۲}B_{۲۱},$$
$$C_{۲۲}=A_{۲۱}B_{۱۲}+A_{۲۲}B_{۲۲}.$$

در نتیجه، به هشت ضرب و چهار جمع روی ماتریس‌های $\frac{n}{۲}\times\frac{n}{۲}$ منجر می‌شود. حال می‌توان مسئله را به صورت بازگشتی بازنوسی کردو از تقسیم و غلبه بهره برد. شبه‌کد آن به صورت زیر است.

```
Zarb_matrix_ToQ(A, B, C, n)
    if n == 1
        C[0,0] = C[0,0] + A[0,0] × B[0,0]
        return
    k = n/2
    A11 = A[0:k-1,0:k-1], A12 = A[0:k-1,k:n-1], A21 = A[k:n-1,0:k-1], A22 =A[k:n-1, k:n-1]
    B11 = B[0:k-1,0:k-1], B12 = B[0:k-1,k:n-1], B21 = B[k:n-1,0:k-1], B22 = B[k:n-1, k:n-1]

    Zarb_matrix_ToQ(A11, B11, C11, k)
    Zarb_matrix_ToQ(A11, B12, C12, k)
    Zarb_matrix_ToQ(A21, B11, C21, k)
    Zarb_matrix_ToQ(A21, B12, C22, k)
    Zarb_matrix_ToQ(A12, B21, C11, k)
    Zarb_matrix_ToQ(A12, B22, C12, k)
    Zarb_matrix_ToQ(A22, B21, C21, k)
    Zarb_matrix_ToQ(A22, B22, C22, k)
```

پس روش بالا هر ماتریس به چهار ماتریس مربع کوچکتر تقسیم می‌شود. همین‌طور ماتریس حاصل C نیز به چهار بخش تقسیم می‌شود. با توجه به معادلات به صورت بازگشتی هر یک از چهار بخش از ضرب و جمع زیرماتریس‌های حاصل از ماتریس‌های ورودی A و B بدست می‌آیند. الگوریتم ضرب ماتریس با تقسیم و غلبه در زبان سی به صورت زیر پیاده شده است.

```
#include <stdio.h>
#include <stdlib.h>


// Takhsis Hafeze jahate matrix
int** takhsisMatrix(int t_radifha, int t_soton) {
   int** matrix = (int**)malloc(t_radifha * sizeof(int*));
   for (int i = 0; i < t_radifha; i++) {
      matrix[i] = (int*)malloc(t_soton * sizeof(int));
   }
   return matrix;
}


// Tabe rahasazi hafezeye eshqali matrix
void rahasaziMatrix(int** matrix, int t_radifha) {
   for (int i = 0; i < t_radifha; i++) {
      free(matrix[i]);
   }
   free(matrix);
}


// Jame do matrixs
void jameMatrix(int** A, int** B, int** C, int andaze) {
   for (int i = 0; i < andaze; i++) {
      for (int j = 0; j < andaze; j++) {
```

```
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}


// tafriqe do matrix
void tfriqMatrix(int** A, int** B, int** C, int andaze) {
    for (int i = 0; i < andaze; i++) {
        for (int j = 0; j < andaze; j++) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}


// Zarb Bazghashti
int** ZarbBazgashti(int** A, int** B, int andaze) {
    if (andaze == 1) {
        int** C = takhsisMatrix(1, 1);
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }


    int andazeJadid = andaze / 2;
    int** C = takhsisMatrix(andaze, andaze);


    // takhsise zirMatrix-ha
    int** A11 = takhsisMatrix(andazeJadid, andazeJadid);
    int** A12 = takhsisMatrix(andazeJadid, andazeJadid);
    int** A21 = takhsisMatrix(andazeJadid, andazeJadid);
    int** A22 = takhsisMatrix(andazeJadid, andazeJadid);


    int** B11 = takhsisMatrix(andazeJadid, andazeJadid);
    int** B12 = takhsisMatrix(andazeJadid, andazeJadid);
    int** B21 = takhsisMatrix(andazeJadid, andazeJadid);
    int** B22 = takhsisMatrix(andazeJadid, andazeJadid);


    // Tqsim matrix-ha be zirmatrixha
    for (int i = 0; i < andazeJadid; i++) {
        for (int j = 0; j < andazeJadid; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + andazeJadid];
            A21[i][j] = A[i + andazeJadid][j];
            A22[i][j] = A[i + andazeJadid][j + andazeJadid];


            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + andazeJadid];
            B21[i][j] = B[i + andazeJadid][j];
```

```
        B22[i][j] = B[i + andazeJadid][j + andazeJadid];
    }
}


// matrix-haye mvqt
int** C11 = takhsisMatrix(andazeJadid, andazeJadid);
int** C12 = takhsisMatrix(andazeJadid, andazeJadid);
int** C21 = takhsisMatrix(andazeJadid, andazeJadid);
int** C22 = takhsisMatrix(andazeJadid, andazeJadid);


int** mvqt1 = takhsisMatrix(andazeJadid, andazeJadid);
int** mvqt2 = takhsisMatrix(andazeJadid, andazeJadid);


// Mohasebe bazgashti matrix-ha
// C11 = A11 * B11 + A12 * B21
int** P1 = ZarbBazgashti(A11, B11, andazeJadid);
int** P2 = ZarbBazgashti(A12, B21, andazeJadid);
jameMatrix(P1, P2, C11, andazeJadid);


// C12 = A11 * B12 + A12 * B22
int** P3 = ZarbBazgashti(A11, B12, andazeJadid);
int** P4 = ZarbBazgashti(A12, B22, andazeJadid);
jameMatrix(P3, P4, C12, andazeJadid);


// C21 = A21 * B11 + A22 * B21
int** P5 = ZarbBazgashti(A21, B11, andazeJadid);
int** P6 = ZarbBazgashti(A22, B21, andazeJadid);
jameMatrix(P5, P6, C21, andazeJadid);


// C22 = A21 * B12 + A22 * B22
int** P7 = ZarbBazgashti(A21, B12, andazeJadid);
int** P8 = ZarbBazgashti(A22, B22, andazeJadid);
jameMatrix(P7, P8, C22, andazeJadid);


// Tarkibe ntayej
for (int i = 0; i < andazeJadid; i++) {
    for (int j = 0; j < andazeJadid; j++) {
        C[i][j] = C11[i][j];
        C[i][j + andazeJadid] = C12[i][j];
        C[i + andazeJadid][j] = C21[i][j];
        C[i + andazeJadid][j + andazeJadid] = C22[i][j];
    }
}


// rahasazi fazaye matirx-ha
rahasaziMatrix(A11, andazeJadid);
```

```
    rahasaziMatrix(A12, andazeJadid);
    rahasaziMatrix(A21, andazeJadid);
    rahasaziMatrix(A22, andazeJadid);
    rahasaziMatrix(B11, andazeJadid);
    rahasaziMatrix(B12, andazeJadid);
    rahasaziMatrix(B21, andazeJadid);
    rahasaziMatrix(B22, andazeJadid);
    rahasaziMatrix(C11, andazeJadid);
    rahasaziMatrix(C12, andazeJadid);
    rahasaziMatrix(C21, andazeJadid);
    rahasaziMatrix(C22, andazeJadid);
    rahasaziMatrix(P1, andazeJadid);
    rahasaziMatrix(P2, andazeJadid);
    rahasaziMatrix(P3, andazeJadid);
    rahasaziMatrix(P4, andazeJadid);
    rahasaziMatrix(P5, andazeJadid);
    rahasaziMatrix(P6, andazeJadid);
    rahasaziMatrix(P7, andazeJadid);
    rahasaziMatrix(P8, andazeJadid);
    rahasaziMatrix(mvqt1, andazeJadid);
    rahasaziMatrix(mvqt2, andazeJadid);


    return C;
}


int main() {
    int n = 4; // Matrix andaze (tavan 2)


    int** A = takhsisMatrix(n, n);
    int** B = takhsisMatrix(n, n);


    // mqdardehi avaleye
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = i + j;
            B[i][j] = i - j;
        }
    }


    // farakhani tabe zarb bazgashti
    int** C = ZarbBazgashti(A, B, n);


    // nemayesh ntayej
    printf("Matrix C:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", C[i][j]);
        }
```

```
    printf("\n");
  }


  // Rahasazi Hafezeye takhsisi
  rahasaziMatrix(A, n);
  rahasaziMatrix(B, n);
  rahasaziMatrix(C, n);


  return 0;
}
```

اما همچنان دارای مرتبه زمانی $O\left(n^{۳}\right)$ است. تمرین- گزاره اخیر را بررسی کنید.

ضرب ماتریسی با روش تقسیم و غلبه- روش استراسن

به عنوان تلاش دیگری از روش تقسیم و غلبه، از روشی که استراسن معرفی کرد بهره می‌گیریم. همانند تلاش قبلی تقسیم و غلبهٔ هر ماتریس به چهار زیرماتریس تقسیم می‌شود.

$$\begin{bmatrix} C_{۱۱} & C_{۱۲} \\ C_{۲۱} & C_{۲۲} \end{bmatrix} = \begin{bmatrix} A_{۱۱} & A_{۱۲} \\ A_{۲۱} & A_{۲۲} \end{bmatrix} \begin{bmatrix} B_{۱۱} & B_{۱۲} \\ B_{۲۱} & B_{۲۲} \end{bmatrix}$$

$$m_{۱} = \left(A_{۱۱} + A_{۲۲}\right)\left(B_{۱۱} + B_{۲۲}\right)$$

$$m_{۲} = \left(A_{۲۱} + A_{۲۲}\right)B_{۱۱}$$

$$m_{۳} = A_{۱۱}\left(B_{۱۲} - B_{۲۲}\right)$$

$$m_{۴} = A_{۲۲}\left(B_{۲۱} - B_{۱۱}\right)$$

$$m_{۵} = \left(A_{۱۱} + A_{۱۲}\right)B_{۲۲}$$

$$m_{۶} = \left(A_{۲۱} - A_{۱۱}\right)\left(B_{۱۱} + B_{۱۲}\right)$$

$$m_{۷} = \left(A_{۱۲} - A_{۲۲}\right)\left(B_{۲۱} + B_{۲۲}\right)$$

با محاسبهٔ مقادیر میانی می‌توان حاصل ماتریس زیر را حساب کرد.

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

تمرین- درستی محاسبات را بررسی کنید. تعداد ضرب و جمع در هر مرحله چه تعداد است. شبه‌کد آن به صورت زیر است.

```
Strassen(A, B, C, n)
    if n == 1
        C[0,0] = C[0,0] + A[0,0] * B[0,0]
        return
    k = n/2
    A11 = A[0:k-1,0:k-1], A12 = A[0:k-1,k:n-1], A21 = A[k:n-1,0:k-1], A22 =A[k:n-1, k:n-1]
    B11 = B[0:k-1,0:k-1], B12 = B[0:k-1,k:n-1], B21 = B[k:n-1,0:k-1], B22 = B[k:n-1, k:n-1]
    m1 = Strassen((A11 + A22), (B11 + B22), k)
    m2 = Strassen((A21 + A22), B11, k)
    m3 = Strassen(A11, (B12 - B22), k)
    m4 = Strassen(A22, (B21 - B11), k)
    m5 = Strassen((A11 + A12), B22, k)
    m6 = Strassen((A21 - A11), (B11 + B12), k)
    m7 = Strassen((A12 - A22), (B21 + B22), k)
    C11 = m1 + m4 - m5 + m7
    C12 = m3 + m5
    C21 = m2 + m4
    C22 = m1 + m3 - m2 + m6
```

کد آن به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


// tabe takhsis hafeze be matrix
int** takhsisMatrix(int t_radifha, int t_soton) {
   int** matrix = (int**)malloc(t_radifha * sizeof(int*));
```

```
    for (int i = 0; i < t_radifha; i++) {
        matrix[i] = (int*)malloc(t_soton * sizeof(int));
    }
    return matrix;
}


// Tabe rahasazi hafezeye eshqali matrix
void rahasaziMatrix(int** matrix, int t_radifha) {
    for (int i = 0; i < t_radifha; i++) {
        free(matrix[i]);
    }
    free(matrix);
}


// Tabe zarb do matrix
int** zarbmatrixha_sade(int** A, int** B, int t_radifhaA, int t_sotonA, int t_sotonB) {
    int** C = takhsisMatrix(t_radifhaA, t_sotonB);
    for (int i = 0; i < t_radifhaA; i++) {
        for (int j = 0; j < t_sotonB; j++) {
            C[i][j] = 0;
            for (int k = 0; k < t_sotonA; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}


// Jame do matrixs
void jameMatrix(int** A, int** B, int** C, int andaze) {
    for (int i = 0; i < andaze; i++) {
        for (int j = 0; j < andaze; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}


// tfriqe do matrix
void tfriqMatrix(int** A, int** B, int** C, int andaze) {
    for (int i = 0; i < andaze; i++) {
        for (int j = 0; j < andaze; j++) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}


// Zarbe matrix-e Strassen
int** strassen(int** A, int** B, int andaze) {
```

```
if (andaze == 1) {
   int** C = takhsisMatrix(1, 1);
   C[0][0] = A[0][0] * B[0][0];
   return C;
}


int andazeJadid = andaze / 2;
int** C = takhsisMatrix(andaze, andaze);


// Takhsise zirmatrix-ha
int** A11 = takhsisMatrix(andazeJadid, andazeJadid);
int** A12 = takhsisMatrix(andazeJadid, andazeJadid);
int** A21 = takhsisMatrix(andazeJadid, andazeJadid);
int** A22 = takhsisMatrix(andazeJadid, andazeJadid);


int** B11 = takhsisMatrix(andazeJadid, andazeJadid);
int** B12 = takhsisMatrix(andazeJadid, andazeJadid);
int** B21 = takhsisMatrix(andazeJadid, andazeJadid);
int** B22 = takhsisMatrix(andazeJadid, andazeJadid);


// tqsime matrix-ha be zirmatrix-ha
for (int i = 0; i < andazeJadid; i++) {
   for (int j = 0; j < andazeJadid; j++) {
      A11[i][j] = A[i][j];
      A12[i][j] = A[i][j + andazeJadid];
      A21[i][j] = A[i + andazeJadid][j];
      A22[i][j] = A[i + andazeJadid][j + andazeJadid];


      B11[i][j] = B[i][j];
      B12[i][j] = B[i][j + andazeJadid];
      B21[i][j] = B[i + andazeJadid][j];
      B22[i][j] = B[i + andazeJadid][j + andazeJadid];
   }
}


// matrix-haye mvqt
int** M1 = takhsisMatrix(andazeJadid, andazeJadid);
int** M2 = takhsisMatrix(andazeJadid, andazeJadid);
int** M3 = takhsisMatrix(andazeJadid, andazeJadid);
int** M4 = takhsisMatrix(andazeJadid, andazeJadid);
int** M5 = takhsisMatrix(andazeJadid, andazeJadid);
int** M6 = takhsisMatrix(andazeJadid, andazeJadid);
int** M7 = takhsisMatrix(andazeJadid, andazeJadid);


int** mvqt1 = takhsisMatrix(andazeJadid, andazeJadid);
int** mvqt2 = takhsisMatrix(andazeJadid, andazeJadid);
```

```
// M1 = (A11 + A22) * (B11 + B22)
jameMatrix(A11, A22, mvqt1, andazeJadid);
jameMatrix(B11, B22, mvqt2, andazeJadid);
M1 = strassen(mvqt1, mvqt2, andazeJadid);


// M2 = (A21 + A22) * B11
jameMatrix(A21, A22, mvqt1, andazeJadid);
M2 = strassen(mvqt1, B11, andazeJadid);


// M3 = A11 * (B12 - B22)
tfriqMatrix(B12, B22, mvqt2, andazeJadid);
M3 = strassen(A11, mvqt2, andazeJadid);


// M4 = A22 * (B21 - B11)
tfriqMatrix(B21, B11, mvqt2, andazeJadid);
M4 = strassen(A22, mvqt2, andazeJadid);


// M5 = (A11 + A12) * B22
jameMatrix(A11, A12, mvqt1, andazeJadid);
M5 = strassen(mvqt1, B22, andazeJadid);


// M6 = (A21 - A11) * (B11 + B12)
tfriqMatrix(A21, A11, mvqt1, andazeJadid);
jameMatrix(B11, B12, mvqt2, andazeJadid);
M6 = strassen(mvqt1, mvqt2, andazeJadid);


// M7 = (A12 - A22) * (B21 + B22)
tfriqMatrix(A12, A22, mvqt1, andazeJadid);
jameMatrix(B21, B22, mvqt2, andazeJadid);
M7 = strassen(mvqt1, mvqt2, andazeJadid);


// tarkib ntayej
for (int i = 0; i < andazeJadid; i++) {
   for (int j = 0; j < andazeJadid; j++) {
      C[i][j] = M1[i][j] + M4[i][j] - M5[i][j] + M7[i][j];
      C[i][j + andazeJadid] = M3[i][j] + M5[i][j];
      C[i + andazeJadid][j] = M2[i][j] + M4[i][j];
      C[i + andazeJadid][j + andazeJadid] = M1[i][j] - M2[i][j] + M3[i][j] + M6[i][j];
   }
}


// Rahasazi fazaye matrix-ha
rahasaziMatrix(A11, andazeJadid);
rahasaziMatrix(A12, andazeJadid);
rahasaziMatrix(A21, andazeJadid);
```

```
    rahasaziMatrix(A22, andazeJadid);
    rahasaziMatrix(B11, andazeJadid);
    rahasaziMatrix(B12, andazeJadid);
    rahasaziMatrix(B21, andazeJadid);
    rahasaziMatrix(B22, andazeJadid);
    rahasaziMatrix(M1, andazeJadid);
    rahasaziMatrix(M2, andazeJadid);
    rahasaziMatrix(M3, andazeJadid);
    rahasaziMatrix(M4, andazeJadid);
    rahasaziMatrix(M5, andazeJadid);
    rahasaziMatrix(M6, andazeJadid);
    rahasaziMatrix(M7, andazeJadid);
    rahasaziMatrix(mvqt1, andazeJadid);
    rahasaziMatrix(mvqt2, andazeJadid);


    return C;
}
int main() {
    int n = 256; // andaze Matrix  (tavan 2)
    int** A = takhsisMatrix(n, n);
    int** B = takhsisMatrix(n, n);


    // mqdardehi avalye
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = (i==j) ? i + 1 : 0; //i + j;
            B[i][j] = (i==j) ? i + 1: 0; //i - j;
        }
    }

    int** C = strassen(A, B, n);
    // chap matrix hasel
    printf("Matrix C:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    // rahaszi hafeze
    rahasaziMatrix(A, n);
    rahasaziMatrix(B, n);
    rahasaziMatrix(C, n);
    return 0;
}
```

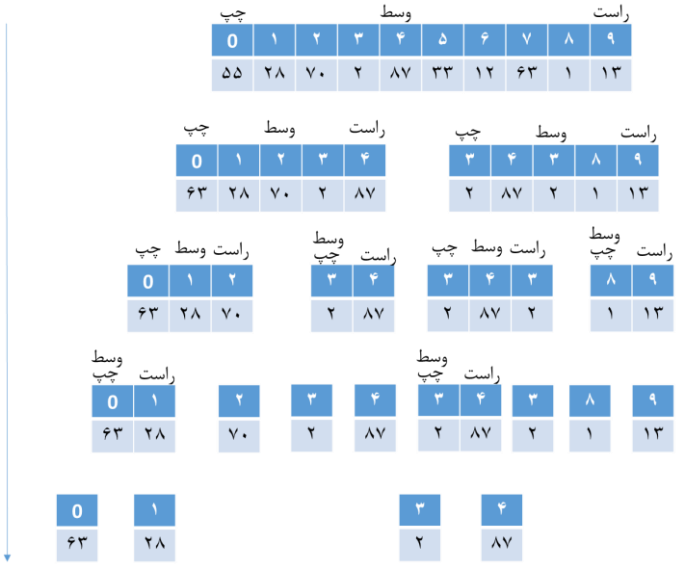
مرتبه زمانی آن عبارت است از

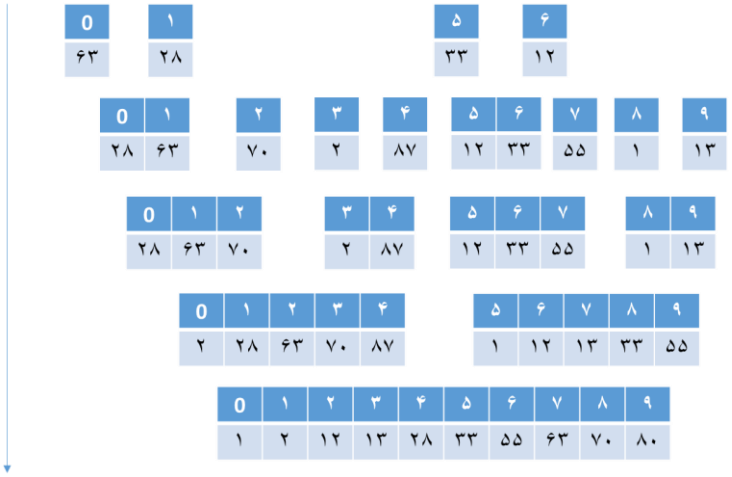
$$z(n) = ۷z\left(\frac{n}{۴}\right) + \theta\left(n^{۲}\right)$$

تمرین- مقدار دقیق ضرب و جمع‌های هر مرحله چقدر است.

مرتب‌سازی ادغامی

بدون فوت وقت طرح اولیهٔ الگوریتم را شرح می‌دهیم. اگر $n = ۱$، آن‌گاه آرایه مرتب است. اگر دو زیرفهرست با اندیس‌های $[0: n/۲]$ و $[n/۲ + ۱: n - ۱]$ جداگانه مرتب صعودی باشند، جهت مرتب‌سازی سراسری کافی است که دو فهرست مذکور ا با یکدیگر ادغام کنیم. می‌توان طرح اولیه را با دو عمل تقسیم و ترکیب پیش برد. شکل زیر مثالی از نحوهٔ ادغام را نشان می‌دهد. در هنگام تقسیم هیچ جابجایی اعضای فهرست انجام نمی‌پذیرد.





شبه‌کد بخش ادغام آن به صورت زیر است.

```
Alg. Edqam(arr, c, v, r)
    arr_c = arr[c:v ];      arr_r[v+1:r]
    i = j = 0
    k = c
    while i <= v & j <= r – v
        if arr_c[i] <= arr_r[j]
            arr[k] = arr_c[i++]
        else
            arr[k] = arr_r[j++]
        k++
    while i <=v
        arr[k] = arr_c[i++]
        i++; k++
    while j <=r - v
        arr[k] = arr_r[j++]
        j++; k++
```

الگوریتم تقسیم و غلبه بر اساس تابع ادغام به صورت زیر تعریف می‌شود.

```
Alg. Morattabsazi_edqami(arr, c, r)
    If c < r
        v = c + (r – c) / 2
    Morattabsazi_edqami(arr, c, v)
    Morattabsazi_edqami(arr, v + 1 r)
    Edqam(arr, c, v, r)
```

با فراخوانی الگوریتم Morattabsazi_edqami(arr, 0, n-1) شبه‌کد بالا آرایه ورودی را مرتب می‌سازد. الگوریتم مرتب‌سازی ادغامی را می‌توان در زبان سی به صورت پیاده کرد.

```
#include <stdio.h>
#include <stdlib.h>


// edqam do zirараye chp arr[chp..vst] o zirараye rast arr[vst+1..rast]
void edqam(int arr[], int chp, int vst, int rast) {
		int i, j, k;
		int n1 = vst - chp + 1;
		int n2 = rast - vst;
		int chpArr[n1], rastArr[n2];
		for (i = 0; i < n1; i++)
			chpArr[i] = arr[chp + i];
		for (j = 0; j < n2; j++)
			rastArr[j] = arr[vst + 1 + j];
		// edqam zirараye
		i = 0;
		j = 0;
		k = chp;
		while (i < n1 && j < n2) {
			if (chpArr[i] <= rastArr[j]) {
				arr[k] = chpArr[i];
				i++;
			}
			else {
				arr[k] = rastArr[j];
				j++;
			}
			k++;
		}


		// Copy mqadir baqimande mojode rastArr[]
		while (i < n1) {
			arr[k] = chpArr[i];
			i++;
			k++;
		}


		// Copy mqadir baqimande mojode rastArr[]
		while (j < n2) {
			arr[k] = rastArr[j];
			j++;
			k++;
		}
}
```

تعریف تابع بازگشتی مرتب‌سازی ادغامی

```
// edqame morattab dar bazeye [chp-rast]
void Morattabsazi_edqami(int arr[], int chp, int rast) {
        if (chp < rast) {
                int vst = chp + (rast - chp) / 2;


                // moratabsazi nimeye aval o dovvom
                Morattabsazi_edqami(arr, chp, vst);
                Morattabsazi_edqami(arr, vst + 1, rast);


                // edqam nimehaye morattab
                edqam(arr, chp, vst, rast);
        }
}


int main() {
        int arr[] = {12, -11, 13, 5, 6, -97};
        int n = sizeof(arr) / sizeof(arr[0]);


        // farakhani tabe
        Morattabsazi_edqami(arr, 0, n - 1);


        // nemayesh arraye morattab
        for (int i = 0; i < n; i++)
                printf("%d ", arr[i]);
        return 0;
}
```

مرتبه زمانی الگوریتم به صورت زیر است.

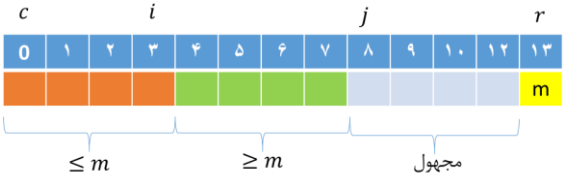
$$z(n) = ۲z\left(\frac{n}{۲}\right) + n, z\left(۱\right) = ۱$$

با استفاده از قضیه اصلیْ زمان اجرای الگوریتم مرتب‌سازی ادغامی از مرتبهٔ $O(nlog\ n)$ است. اگر از روش درختی حل کنیم به مقدار $nlogn + n$ یا $\theta(nlog\ n)$ خواهیم رسید.

مرتب‌سازی سریع

الگوریتم مرتب‌سازی ادغامی به نحوی است که در ابتدا آرایه به زیرآرایه‌ها تقسیم و سپس یک به یک مرتب و در نهایت با یکدیگر ادغام می‌شوند. روش حل مرتب‌سازی سریع بر این پرسش استوار است که می‌توان الگوریتم را به گونه‌ای طراحی کرد که تعداد مراحل را کاهش داد. مرتب‌سازی حین تقسیم به دو زیرآرایه به گونه‌ای است که دیگر نیازی به ادغام فهرست‌های مرتب نباشد. به دیگر سخن، آرایه چنان تقسیم شود که مقادیر قسم سمت چپ از مقادیر قسم سمت راست کوچکتر باشند. الگوریتم و افراز

مذکور را جهت مرتب‌سازی «هوور» معرفی کرد که افرازی درجا است. در ابتدا تابع افراز را بررسی می‌کنیم. هر بار عددی از آرایه به عنوان محور انتخاب می‌شود و سعی می‌شود در جایگاه درست خود در آرایه قرار گیرد. به دیگر سخن، با انتخاب محوری در آغاز دور در انتهای دور مذکور، مقادیر کوچکتر قبل محور و مقادیر بزرگتر در سمت راست آن قرار گیرد. شکل زیر بخشی از تنظیم اعداد را نسبت به محور با مقدار m نشان می‌دهد.



پس طراح کلی الگوریتم شامل دو مرحلهٔ زیر است.

تقسیم- افراز آرایه به دو آرایه که در بخش چپ مقادیر کوچکتر از یا مساوی با محور و در بخش راست بزرگتر از محور هستند.

غلبه- فراخوانی بازگشتانهٔ مرتب سریع جهت مرتب‌کردن زیرفهرست‌ها

- ترکیب عدم انجام کاری! دلیل؟ زیرا آرایه‌ها خودبخود مرتب هستند.

شبه‌کد مرتب‌سازی سریع به صورت زیر است.

```
Alg. Efraz(arr, c, r)
    m = arr[r]
    int i = c -1,
    for j = c: r-1
        if arr[j] <= m
            i++
            arr[i]↔arr[j]
    arr[i + 1]↔arr[r]
  return i + 1
Alg. Morattabsazi_sarie(arr, c, r)
    if c < r
        q = Efraz(arr, c, r)
        Morattabsazi_sarie(arr, c, q - 1)
        Morattabsazi_sarie(arr, q + 1, r)
```

همان‌طور که مشخص است،الگوریتم از نوع تقسیم و غلبه است. تمرین نشان دهید گزاره درست است.

تعریف تابع بازگشتی مرتب‌سازی سریع در زبان سی به صورت زیر می‌تواند باشد.

```
#include <stdio.h>
// Tabe efraz
int efraz(int arr[], int c, int r) {
   int m = arr[r]; // entekhab mehvar (m)
   int i = c - 1;
   for (int j = c; j < r; j++) {
      if (arr[j] < m) { // varesi kochetar bodan ozve kononi az m
         i++;
         // jabejaee
         int mvqt = arr[i];
         arr[i] = arr[j];
         arr[j] = mvqt;
      }
   }
   // Jabejaee ozve m ba ozve dar andise i+1
   int mvqt = arr[i + 1];
   arr[i + 1] = arr[r];
   arr[r] = mvqt;
   return i + 1; // bargardandan mqdare andise efraz
}
// Tabe morattabzasi sarie
void morattabsazi_sarie(int arr[], int c, int r) {
   if (c < r) {
      int j = efraz(arr, c, r); // andis efraz
      // morattabsazi bazgashti pishe o pase mhvar
      morattabsazi_sarie(arr, c, j - 1);
      morattabsazi_sarie(arr, j + 1, r);
   }
}
int main() {
   int arr[] = {12, -11, 13, 5, 6, -97};
   int n = sizeof(arr) / sizeof(arr[0]);
   morattabsazi_sarie(arr, 0, n - 1);
   printf("\n array morattab: ");
   for(int i = 0; i < n; i++)  {
      printf(" %4d", arr[i]);
   }
   printf("\n");
   return 0;
}
```

تحلیل پیچیدگی الگوریتم مرتب‌سازی سریع بدین صورت است که بهترین زمان اجرا هنگامی رخ می‌دهد که آرایه به دو زیرآرایهٔ برابر تقسیم شود (چرا؟).

$$z(n) = ۲z\left(\frac{n}{۲}\right) + n, z\left(۱\right) = ۱$$

با استفاده از قضیه اصلیْ مرتبهٔ الگوریتم $O(nlog\ n)$ است. بدترین زمان هم هنگام تقسیم آرایه به دو زیرآرایه با اندازه‌های $n - ۱$ و ۱ است (چرا؟).

$$z(n) = z\left(n - ۱\right) + z\left(۱\right) + n = z\left(n - ۱\right) + \theta(n)$$

با استفاده از قضیه اصلیْ مرتبه $O(n^۲)$ خواهد بود.

زمان میانگین را می‌توان در قالبی مثالی بدست آورد. فرض می‌کنیم هر بار الگوریتمْ آرایه را به دو زیر آرایهٔ با نسبت ۴ به ۱ تقسیم می‌کند.

$$z(n) = z\left(\frac{۴n}{۵}\right) + z\left(\frac{n}{۵}\right) + n$$

طبق روش درختی محاسبه زمان اجرا میانگین برابر $O(nlog\ n)$ است.

تمرین- الگوریتم مرتب‌سازی سریع را به نحوی تغییر دهید که محور همیشه از ابتدا انتخاب شود. اندیس i از ابتدای الگوریتم و اندیس j از انتهای آرایه شروع به حرکت کند.

مرتب‌سازی سریع تصادفی

الگوریتم مرتب‌سازی تصادف بستگی به انتخاب محور دارد. پس برای آرایه‌های نسبتا مرتب ممکن است نتیجه‌ای مناسب نداشته باشد. یکی از راه‌حل‌ها انتخاب تصادفی محور است که معروف به مرتب‌سازی سریع تصادفی است. تعریف افراز تصادفی و تابع مرتب‌سازی سریع تصادفی را در کد زیر می‌بینید.

```
Alg. Efraz_tsadfi(arr, c, r)
    t = adadi tasdofi ∈{c,…,r}
    arr[t]⟷arr[r]
    m = arr[r]
    int i = c -1,
    for j = c: r-1
        if arr[j] <= m
            i++
            arr[i]⟷arr[j]
    arr[i + 1]⟷arr[r]
   return i + 1
Alg. Morattabsazi_sarie(arr, c, r)
    if c < r
```

```
        q = _tsadfi (arr, c, r)
        Morattabsazi_sarie(arr, c, q - 1)
        Morattabsazi_sarie(arr, q + 1, r)
```

الگوریتم مرتب‌سازی سریع تصادفی در سی به صورت زیر می‌توان پیاده کرد.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


// Tabe efraz
int efraz_tasadofi(int arr[], int c, int r) {
   // entekhab mhvar tsadofi o jabejaee an ba ozve akhar
   int t = c + rand() % (r - c + 1);
                int mvqt = arr[r];
   arr[r] = arr[t];
   arr[t] = mvqt;
                int m = arr[r]; // entekhab mehvar (m)
   int i = c - 1;
   for (int j = c; j < r; j++) {
      if (arr[j] < m) { // varesi kochetar bodan ozve kononi az m
         i++;
         // jabejaee
         int mvqt = arr[i];
         arr[i] = arr[j];
         arr[j] = mvqt;
      }
   }
   // Jabejaee ozve m ba ozve dar andise i+1
   mvqt = arr[i + 1];
   arr[i + 1] = arr[r];
   arr[r] = mvqt;
   return i + 1; // bargardandan mqdare andise efraz
}
// Tabe morattabzasi sarie
void morattabsazi_sarie(int arr[], int c, int r) {
   if (c < r) {
      int j = efraz_tasadofi(arr, c, r); // andis efraz
      // morattabsazi bazgashti pishe o pase mhvar
      morattabsazi_sarie(arr, c, j - 1);
      morattabsazi_sarie(arr, j + 1, r);
   }
}
int main() {
   int arr[] = {12, -11, 13, 5, 6, -97};
   int n = sizeof(arr) / sizeof(arr[0]);
   morattabsazi_sarie(arr, 0, n - 1);
   for(int i = 0; i < n; i++)  {
      printf(" %4d", arr[i]);
   }
   printf("\n");
```

```
    return 0;
}
```

مسائل بدون امکان راه‌حل تقسیم و غلبه‌ای

سخن کوتاه، تقسیم و غلبه بر حول بازگشتی بودن تابع می‌چرخد. الگوریتمی بازگشتی z(n) است اگر به ازای $n_0 > 0$ ویژگی‌های زیر برقرار باشند:

- $\forall n < n_0 \Longrightarrow z\left(۱\right) = \theta(۱)$
- $\forall n \geq n_0$ هر مسیر بازگشتی با تعدادی متناهی از تکرار به حالت پایهٔ تعریف شده ختم شود.

مسائل دارای ویژگی‌های زیر با روش تقسیم و غلبه دارای راه‌حل نیستند.

- در صورتی که اندازه n به دو یا چند بخش با اندازه تقریبا برابر با n بخش شوند.
- اگر نمونه با اندازه n به n نمونه با اندازه n/c و c ثابت تقسیم شوند. مثال آن فیبوناتچی است. هرچند، گاهی اوقات راه دیگری نداریم. برج هانوی از چنین موارد است.