۱ – مرتب‌سازی سریع[1]

مرتب‌سازی سریع یک الگوریتم مرتب‌سازی است که در بدترین حالت $\Theta(n^2)$ است. با وجود کندی بدترین حالت مرتب‌سازی سریع، مرتب‌سازی سریع اغلب بهترین انتحاب عملی است، چون در حالت میانگین به طور قابل توجهی کارا است؛ زمان مورد انتظار برای اجرا $\Theta(nlgn)$ است.
مرتب‌سازی سریع همچنین مزیت مرتب کردن درجا را هم دارا است.
ضریب ثابت nlgn کاملا کوچک است و این الگوریتم برای حافظه‌های خارجی و موازی نیز کارا می‌باشد.

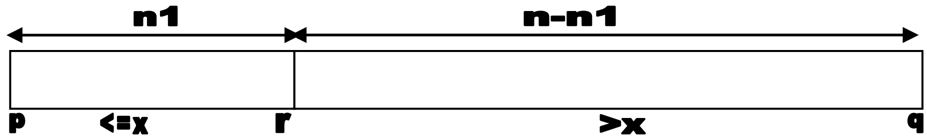
۱ – ۱ – توصیف مرتب‌سازی سریع

مرتب‌سازی سریع، مانند مرتب‌سازی ادغامی[2]، بر اساس مدل Divide & Conquer بنا شده است. در ادامه سه مرحله لازم برای مرتب سازی یک آرایه‌ی A[p...q] آورده شده است.

Divide : partition کردن آرایه‌ی A[p...q] به دو زیرآرایه‌ی A[p...r] و A[r+1...q] به گونه‌ای که تمامی عناصر زیرآرایه‌ی اول کوچکتر یا مساوی x و تمامی عناصر زیرآرایه دوم بزرگتر از x هستند.

Conquer : مرتب کردن زیرآرایه‌های A[p...r] و A[r+1...q] با فراخوانی بازگشتی مرتب‌سازی سریع.

Combine: چون زیرآرایه‌ها درجا مرتب شده‌اند نیازی به ادغام آن‌ها نیست و کل آرایه‌ی A[p...q] اکنون مرتب است.



شکل ۱ - دو ناحیه‌ی به‌وجود آمده توسط partition روی آرایه‌ی A[p...q].
مقادیرA[p...r] کوچکتر یا مساوی x و مقادیA[r+1...q] بزرگتر از x هستند.

---

[1] -Quick Sort

[2] -Merge Sort

به طور کلی می‌توان چهار حالت را برای انتخاب محور[1] x نام برد:

۱. مرتب‌سازی سریع ساده: در این روش محور x اولین عنصر آرایه است.

۲. مرتب‌سازی سریع تصادفی: در این روش محور xیکی از عناصر آرایه به صورت تصادفی است.

۳. مرتب‌سازی سریع میانگین: در این روش محور x میانگین عناصر آرایه است.

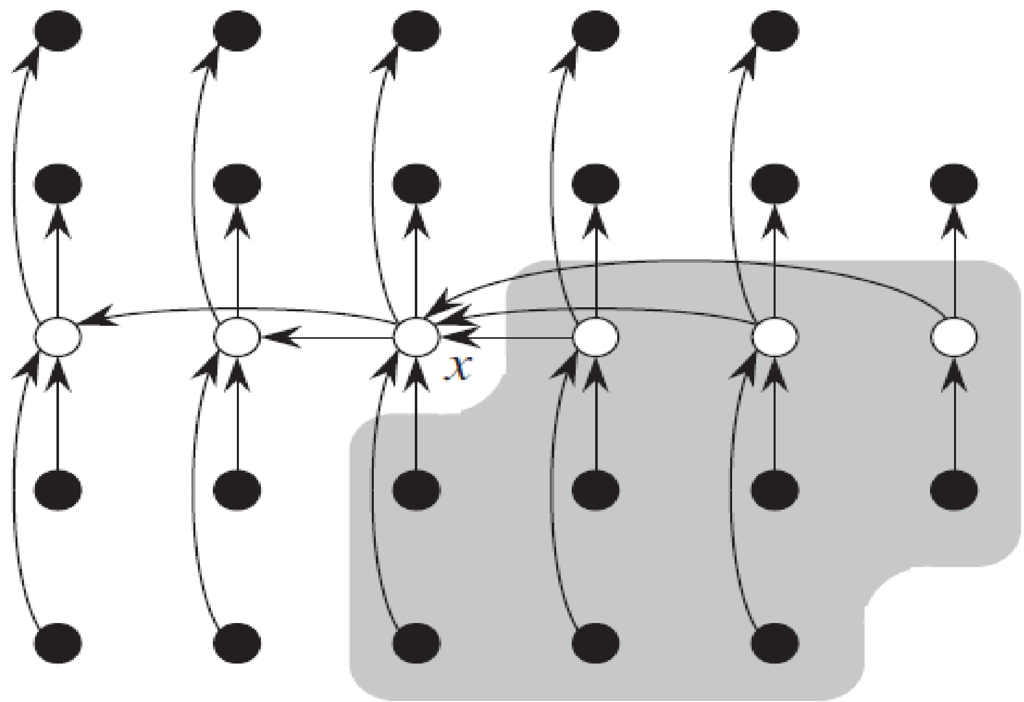
۴. مرتب‌سازی سریع خطی زمانی.

می‌خواهیم با O(n) یک محور انتخاب کنیم که در بدترین حالت نیز مرتب‌سازی سریع هزینه زمانی O(nlgn) داشته باشد

۱.آرایه را به $\lceil n/5 \rceil$ دسته ۵ تایی تقسیم می‌کنیم

۲.هرگروه را با insertion sort مرتب می‌کنیم و میانه ۵ عنصر انتخاب شود, $\lceil n/5 \rceil$ میانه داریم

۳.میانه این $\lceil n/5 \rceil$ عدد محاسبه شود. عدد انتخاب شده محور است.



شکل ۲- انتخاب محورx

---

[1] -Pivot

نتیجه می‌گیریم که:

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil - 2\right) \ \geq\ \frac{3n}{10} - 6\ .$$

ملاحظه می‌شود که یک نسبت از n به دست آمد.(۳/۱۰)

ملاحظه می‌شود که هزینه زمانی آن برابر است با

T(n) = (n/5) * O(1) + O(n) = O(n)

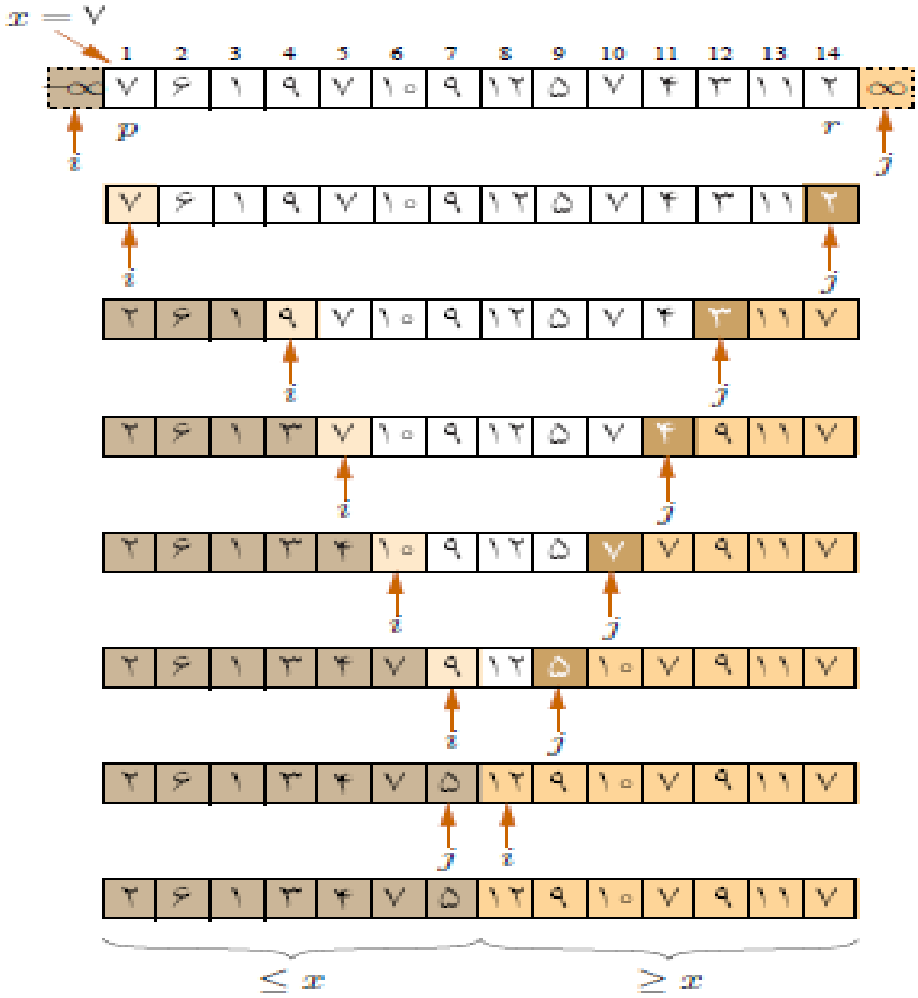
پروسه‌ی زیر نحوه‌ی پیاده‌سازی مرتب‌سازی سریع را برای آرایه‌ی A[p...r] نشان می‌دهد:
partition اندیس اخرین عنصر سمت چپ را بر می‌گرداند.

```
QuickSort(A, p, r)
{
        if (p>= r)      return;
        q = Partition(A, p, r);
        QuickSort(A, p, q);
        QuickSort(A, q+1, r);
}
```

پروسه‌ی زیر نحوه‌ی پیاده‌سازی تابع Partition را برای آرایه‌ی A[p...r] و محور x نشان می‌دهد.

ساده‌ترین روش استفاده از یک ارایه کمکی به طول n می‌باشد (درجا نیست.) روش بهتر استفاده از دو پوینتر است.

```
Partition (A, p, r) {
        x = get_pivot(A, p, r);
        i=p;
        j=r;
        while (j>=i) {
                while (A[i]<=x && i<=j)
                        i++;
                while (A[j]>x && i<=j)
                        j--;
                if (i<j)
                        swap (A[i], A[j]);
        }
        return j;
}
```



شکل ۳- مثالی از partition.

مثال: ارایه زیر را با انتخاب x=۸ به عنوان محور partition کنید.

15	10	7	3	8	3	10	4

حل:

4	10	7	3	8	3	10	15
4	3	7	3	8	10	10	15

۱ ۴ – کارایی مرتب‌سازی سریع

زمان اجرای مرتب‌سازی سریع به نحوه partition کردن بستگی دارد که این به نوبه‌ی خود به این که کدام عنصر به عنوان محور برای partition کردن انتخاب شده بستگی دارد.

۱ ۴ ۱ – حالت کلی

$$T(n) = T(n_1) + T(n-n_1) + \Theta(n)$$

۱ ۴ ۲ – بدترین حالت

بدترین حالت زمانی اتفاق می‌افتد که x بزرگترین یا کوچکترین عنصر باشد آنگاه $n_1=1$ یا $n_1=n-1$ است..

$$\begin{aligned} T(n) &= T(n-1) + T(1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ T(n) &= \Theta(n^2) \end{aligned}$$

۱ ۴ ۳ –بهترین حالت

بهترین حالت زمانی اتفاق می‌افتد که با partition اندازه‌ی هیچ‌کدام از زیرآرایه‌ها از $n/2$ بیشتر نشود؛ یعنی بهتر است محور x میانه‌ی اعداد داخل آرایه انتخاب شود. در این حالت مرتب‌سازی سریع، بسیار سریع‌تر عمل می‌کند.

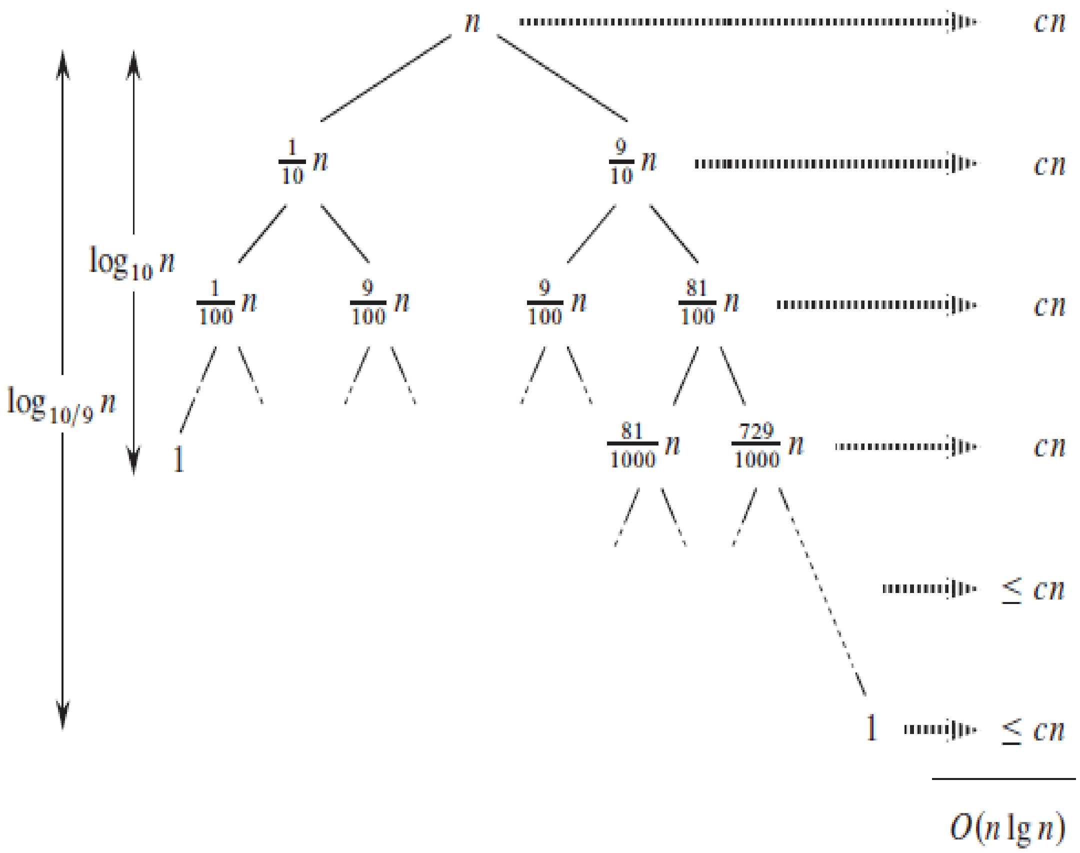
$$T(n) <= 2T(n/2) + \Theta(n)$$
$$T(n) = O(n\lg n)$$

۱ ۴ ۴ –حالت متوازن

حالت متوسط زمان اجرای مرتب‌سازی سریع به بهترین حالت بسیار نزدیک‌تر است تا به بدترین حالت.

فرض کنید، به عنوان مثال، عمل partition همیشه زیرآرایه‌هایی به نسبت ۱ به ۹ تولید می‌کند که در نگاه اول به نظر نامتوازن می‌آید.

$$T(n) <= T(9n/10) + T(n/10) + cn$$
$$T(n) = O(n\lg n)$$



شکل ۴- درخت بازگشتی برای مرتب‌سازی سریع که در آن partition دو قسمت به نسبت ۹به۱ ایجاد می‌کند که زمان اجرا O(nlgn) را نتیجه می‌دهد.

در واقع حتی تقسیم به نسبت ۱ به ۹۹ هم منجر به زمان اجرای O(nlgn) خواهد شد. علت آن است که تقسیم با نسبت ثابت منجر به یک درخت بازگشت با ارتفاع Θ(lgn) می‌شود که هزینه‌ی هر سطح O(n) است. بنابراین زمان اجرا O(nlgn) خواهد بود.

مرتب‌سازی سریع در جایی که تقسیم به نسبت باشد خیلی بهینه است، ولی اگر به دو بخش با اختلاف عددی تقسیم شود بهینه نیست.

۱ ۴ – درجا و پایدار بودن مرتب سازی سریع

درجا	پایدار
خیر	بلی
بلی	خیر

مثال:

اعداد زیر را به روش مرتب‌سازی سریع به صورت صعودی مرتب کنید, اولین عنصر هر آرایه را به عنوان محور در نظر بگیرید.

۸۳۰-۲۶۴-۵۳۸-۱۲۲-۲۹۳-۸۹۲-۱۲۳-۲۳۷-۹۱۰

حل:

(محور)

مثال:

کدام یک از الگوریتم های مرتب‌سازی زیر در شرایطی که آرایه از همان ابتدا به صورت صعودی مرتب شده باشد, بدترین زمان اجرا را دارند؟

Insertion sort	Heap sort	Quick sort	Merge sort

حل:

هزینه ی الگوریتم های Heap Sort و Merge Sort در بدترین و بهترین حالت، هر دو، $O(n\log n)$ می باشد.

هزینه ی الگوریتم Insertion Sort هنگامی که آرایه ی ورودی از همان ابتدا Sort شده باشد، در بهترین حالت و $O(n)$ است.

هزینه ی Quick Sort نیز در این حالت به بدترین حالت خود، یعنی $O(n^2)$ می رسد. زیرا در حالتی که آرایه ی ورودی از همان ابتدا مرتب شده باشد، عمل Partitioning آرایه را به دو قسمت n-1 تایی و 0 تایی تقسیم می کند و در نتیجه هزینه ی زمانی از عبارت $T(n)= T(n-1) + T(0) + \theta(n)$ به دست می آید. ($\theta(n)$ مربوط به هزینه ی عمل Partitioning است.)

پس در بین الگوریتم های داده شده پاسخ صحیح Quick Sort می باشد.

مثال:

در یک آرایه به طول n عددی وجود دارد که بیشتر از n/2ام بار در این آرایه تکرار شده است. الگوریتمی ارائه دهید که این عدد را پیدا کند. الگوریتم شما باید از مرتبه زمانی $O(n)$ و مرتبه حافظه $O(1)$ باشد.

حل:

روی اعداد الگوریتم مرتب‌سازی سریع را انجام می‌دهیم. با این تفاوت که هر دفعه دسته‌ی کوچکتر را دور می‌ریزیم و الگوریتم را روی دسته‌ی بزرکتر انجام می‌دهیم, تا جایی که تمام عناصر موجود در آرایه یکی شوند. این عنصر, عنصر مورد نظر است.

مثال:

پایداری الگوریتم مرتب‌سازی سریع را برای پیاده‌سازی استاندارد آن بررسی کنید.

حل:

Quick Sort : unstable است. مثلا حالتی را در نظر بگیرید که جابجایی نهایی در مرحله ی partition، pivot را از انتهای راست به وسط آرایه می آورد و عنصری را جابجا می کند که ممکن است در اثر این جابجایی، ترتیب این عنصر و عناصر برابر با آن به هم بریزد.

مثال:

نشان دهید چگونه می‌توان هر الگوریتم مرتب‌سازی را به صورت بهینه تغییر داد به گونه‌ای که پایدار شود و این تغییرات چه مقدار زمان و فضای اضافه نیاز دارد.

حل:

همه ی این الگوریتم ها می توانند با شمای زیر Stable شوند، اگر به جای مرتب کردن آرایه ی a1, a2, … , an ، آرایه ی زیر را مرتب کنیم:

(a1,1) , (a2,2) , … , (an, n)

و حالا که هر عنصر دو کلید دارد، هنگامی که $a_i = a_j$ ، a_i از a_j در خروجی پیشی می گیرد، اگر و تنها اگر i < j باشد.

این روش یک مقدار فضای اضافی ثابت به ازای هر عنصر و همچنین یک مقدار زمان ثابت هم به ازای هر مقایسه می گیرد.

پس زمان یا فضای asymptotic را تغییر نمی دهد. (در بدترین حالت زمان و فضای لازم را دو برابر می کند.)

تمرین:

داده‌های زیر را به کمک الگوریتم Quick sort مرتب کنید.

۷-۴-۲-۸-۹-۱-۳-۵

تمرین:

در مورد زمان اجرای partition که روی زیرآرایه‌ای به اندازه n ,$\Theta(n)$ است مختصرا بحث کنید.

تمرین:

همانطور که ادعا شد با استفاده از روش جایگذاری ثابت کنید که $T(n)= \Theta(n^2)$ جواب رابطه بازگشتی $T(n)=T(n-1)+ \Theta(n)$ است.

تمرین:

Quick sort را طوری تغییر دهید که آرایه را ب ترتیب غیرصعودی مرتب کند.

تمرین:

زمان اجرای Quick sort هنگامی که همه عناصر آرایه یک مقدار دارند چیست؟

تمرین:

نشان دهید که زمان اجرای Quick sort هنگامی که همه عناصر آرایه A متفاوت بوده و با ترتیبی نزولی ذخیره شده‌اند, $\Theta(n^2)$ است.

تمرین:

یک ورودی مثال بزنید که مرتب‌سازی سریع روی آن نیازمند $\Omega(n^2)$ مقایسه باشد.محور این مرتب‌سازی را میانه‌ی عنصرهای نخستو میانی و پایانی دنباله بگیرید.

۲ – الگوریتم k-selection

الگوریتم k-selection الگوریتمی برای انتخاب kامین کوچکترین عنصر از یک آرایه است. از این الگوریتم برای پیدا کردن مینیمم(k=0)، ماکزیمم(k=n) و میانه[1](k=n/2) استفاده می‌شود.

۲ ۱ – یافتن k امین عنصر در O(n)

محور را طوری انتخاب می‌کنیم تا تضمین کنیم که اندازه دو بخشدر بدترین حالت O(n) هستند.
برای انتخاب عنصر مورد نظر می‌توان یکی از کارهای زیر را انجام داد:

1.ساده ترین راه ابتدا آرایه را مرتب میکنیم و سپس اندیس kام را برمی‌گردانیم. O(nlgn)

2. kبار مینیمم گرفت. O(kn)

3. ساخت minHeap. (O(n)) و kبار deleteMin. (O(klgn)) [در مجموع O(n+ klgn)]

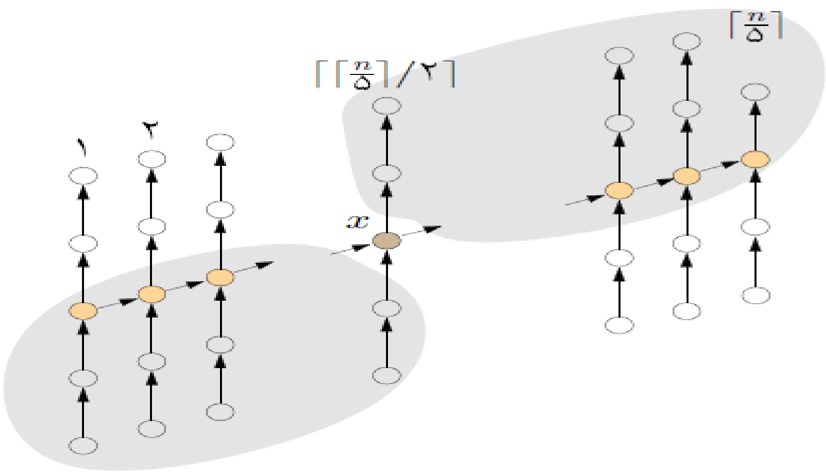
4. استفاده از selection tree; با O(n) مینیمم انتخاب می‌شود, حال kبار مینیمم میگیریم. (O(klgn)) [در مجموع O(n+klgn)]

5. استفاده از partition. (i مقدار بازگشتی تابع partition است.)

اگر k=i عنصر iام عنصر مورد نظر است.

اگر k<i بود به صورت بازگشتی،kامین عنصر سمت چپ را برمی‌گردانیم.

اگر k>i بود به صورت بازگشتی، k-iامین عنصر سمت راست را برمی‌گردانیم.



شکل ۶- انتخاب محور x

---

[1] -Median

حداقل $\left\lceil \frac{\lceil n/۵ \rceil}{۲} \right\rceil - ۲$ گروه دارای ۳ عنصر کوچک‌تر از x هستند.

تعداد عناصر کوچک‌تر از x حداقل

$$۳\left(\left\lceil \frac{۱}{۲}\left\lceil \frac{n}{۵}\right\rceil\right\rceil - ۲\right) \geq \frac{۳n}{۱۰} - ۶$$

است.

به‌صورت مشابه، تعداد عناصر بزرگ‌تر از x نیز حداقل $۳n/۱۰ - ۶$ خواهد بود.

پس، در بدترین حالت، الگوریتم به‌صورت بازگشتی برروی حداکثر $۷n/۱۰ + ۶$ عنصر اعمال خواهد شد.

$$T(n) \leq \begin{cases} \Theta(۱) & n \leq ۸۰ \\ T(\lceil n/۵ \rceil) + T(۷n/۱۰ + ۶) + O(n) & n > ۸۰ \end{cases}$$

فرض: برای یک c مفروض و هر $n \leq ۸۰$ داشته باشیم $T(n) \leq cn$.

$$\begin{aligned} T(n) &\leq c\lceil n/۵ \rceil + c(۷n/۱۰ + ۶) + O(n) \\ &\leq cn/۵ + c + ۷cn/۱۰ + ۶c + O(n) \\ &\leq ۹cn/۱۰ + ۷c + O(n) \\ &\leq cn \end{aligned}$$

پروسه‌ی زیر نحوه‌ی پیاده‌سازی الگوریتم k-select را نشان می‌دهد:

```
kSelect (A, p, r, k)

{

    if(p==r)      return A[p];
    q = Partition (A, p, r);
    if(k==q-p+1)       return A[q];
    if (q-p+1 > k)
            return kSelect(A, p, q-1, k);
    return kSelect (A, q+1, r, k-(q-p+1) );
}
```

۲ ۴ – آنالیز الگوریتم k-select

آنالیز الگوریتم k-select مستقل از k است.

۲ ۴ ۱ –بهترین حالت

$n_1 = n/2$
$T(n) = O(n) + T(n/2)$
$T(n) = O(n)$

۲ ۴ ۲ –بدترین حالت

$n_1 = n-1$ یا $n_1 = 1$
$T(n) = O(n) + T(n-1)$
$T(n) = O(n^2)$

۲ ۴ ۳ –حالت متوسط

$T(n) = O(n)$

مثال:

الگوریتمی ارائه دهید که در کمترین زمان ممکن max و min آرایه‌ای به طول n را حساب کند, هزینه‌ی الگوریتم خود را محاسبه کنید و تعداد کمترین مقایسه لازم را به دست آورید.

حل:

آرایه را به دو قسمت تقسیم می‌کنیم و مینیمم و ماکزیمم را در هر دو زیرآرایه حساب کرده و مقایسه می‌کنیم.

$T(n) = 2T(n/1) + 2\ ,\ \ T(2)=1$
$T(n) = 3n/2 - 2$

$\left\lceil \frac{3n}{2} - 2 \right\rceil$ مقایسه لازم است.

مثال:

الگوریتمی ارائه دهید که در کمترین زمان ممکن عنصر میانه را در یک لینک لیست به دست آورد. هزینه‌ی الگوریتم خود را محاسبه کنید.(توجه داشته باشید که میانه ممکن است میانگین دو عنصر وسط باشد.)

حل:

اگر تعداد اعضا فرد باشد با استفاده از الگوریتمi-select می‌توان عنصر وسط را پیدا کرد و اگر زوج بود با دو بار استفاده از این الگوریتم می‌توان دو عنصر وسط را یافت و میانگین آن ها را بدستآورد. که این الگوریتم $O(n)$ طول می‌کشد. استفاده کردن از لینک لیست تاثیری بر سرعت الگوریتم ندارد و تنها به جای جابجایی عناصر باید اشاره‌گرهای آن‌ها را عوض کرد.

تمرین:

نشان دهید که دومین عنصر کوچک از بین n عضو می‌تواند با $n+\lceil lgn \rceil - 2$ مقایسه در بدترین حالت پیدا شود.(راهنمایی:کوچکترین عضو را نیز پیدا کنید.)

تمرین:

نشان دهید $\lceil 3n/2 \rceil - 2$ مقایسه در بدترین حالت برای پیدا کردن مینمم و ماکزیمم n عدد لازم است.(راهنمایی: در نظر بگیرید که چند عدد به صورت بالقوه مینمم یا ماکزیمم هستند و بررسی کنید که چطور یک مقایسه بر این تعداد تاثیر می‌گذارد.)

تمرین:

در الگوریتم k-select عناصر ورودی به گروه‌های ۵ عنصری تقسیم می‌شوند. آیا اگر عناصر ورودی به گروه‌های ۷ عنصری تقسیم شوند, الگوریتم در زمان خطی کار خواهد کرد؟ ثابت کنید که اگر از گروه‌های ۳ عنصری استفاده شود, الگوریتم در زمان خطی اجرا نمی‌شود.

تمرین:

نشان دهید چگونه مرتب‌سازی سریع می‌تواند در بدترین حات در زمان O(nlgn) اجرا شود.

تمرین:

الگوریتمی با زمان O(n) شرح دهید که , با دریافت مجموعه S با n عدد متفاوت و یک عدد صحیح مثبت k, k<=n عدد در S که به میانه نزدیکترین هستند را مشخص کند.

تمرین:

فرض کنید X[1..n] و Y[1..n] دو آرایه هستند , که هر یک شامل n عدد مرتب شده است. الگوریتمی با زمان O(lgn) ارائه دهید تا میانه تمام 2n عنصر در آرایه X و Y را پیدا کند.