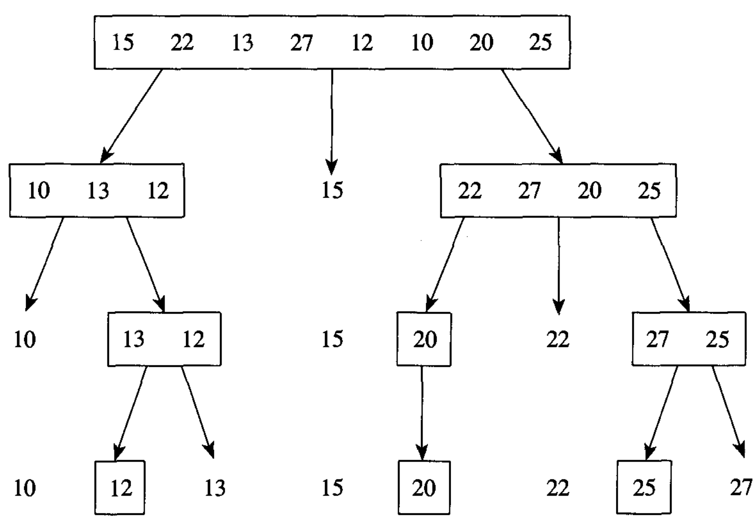
مرتب سازی سریع

توسط Tony Hoare در 1960 ابداع شد که اگر خوب پیاده سازی شود 2تا 3 بار از mergesort و heapsort سریعتر است.

در این روش تقسیم آرایه به دو زیرآرایه طوری انجام می شود که نیاز به ادغام دو زیر آرایه مرتب شده نباشد. در این روش یک عنصر به عنوان محور انتخاب و عناصر کوچکتر از محور در یک آرایه و عناصر بزرگتر از محور در آرایه دیگری ذخیره می شوند (افراز). سپس مرتب سازی سریع بطور بازگشتی فراخوانی می شود تا هر یک از دو آرایه را مرتب کند.

مثال: عنصر اول به عنوان محور:



الگوریتم اصلی:

```
void quicksort (index low, index high)
{
    index pivotpoint;
    if (high > low) {
        partition(low, high, pivotpoint);
        quicksort(low, pivotpoint - 1);
        quicksort(pivotpoint + 1, high);
    }
}
```

الگوریتم افراز

```
void partition (index low, index high,
                index& pivotpoint)
{
    index i, j;
    keytype pivotitem;
    pivotitem = S[low];                    // Choose first item for pivotitem.
    j = low;
    for (i = low + 1; i <= high; i++)
        if (S[i] < pivotitem) {
            j++;
            exchange S[i] and S[j];
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint];     // Put pivotitem at pivotpoint.
}
```

i	j	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	
—	—	15	22	13	27	12	10	20	25	←Initial values
2	1	**15**	**22**	13	27	12	10	20	25	
3	2	**15**	22	**13**	27	12	10	20	25	
4	2	**15**	$\boxed{13}$	$\boxed{22}$	**27**	12	10	20	25	
5	3	**15**	13	22	27	**12**	10	20	25	
6	4	**15**	13	$\boxed{12}$	27	$\boxed{22}$	**10**	20	25	
7	4	**15**	13	12	$\boxed{10}$	22	$\boxed{27}$	**20**	25	
8	4	**15**	13	12	10	22	27	20	**25**	
—	4	$\boxed{10}$	13	12	$\boxed{15}$	22	27	20	25	←Final values

بدترین حالت: زمانی اتفاق می افتد که یک زیر آرایه خالی و دیگری شامل همه عناصر به غیر از عنصر محوری است. داریم

$$T(n) = T(0) + T(n-1) + n - 1, T(0) = 0$$

جواب صریح این رابطه بازگشتی

$$T(n) = \frac{n(n-1)}{2}$$

برای اثبات ادعای بالا کافیست نشان دهیم اگر عنصر محوری دلخواه باشد آنگاه در رابطه بازگشتی کلی زیر

$$T(n) = T(p-1) + T(n-p) + n - 1$$

$1 \le p \le n$

داریم

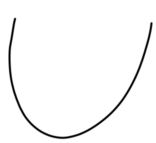
$$T(n) \le \frac{n(n-1)}{2}$$

استقرا

$$T_n \le \underbrace{\frac{(p-1)(p-2)}{2} + \frac{(n-p)(n-p-1)}{2}}_{\text{تابعی از } p,\ 1 \le p \le n} + n-1 \le \frac{(n-1)(n-2)}{2} + n-1$$

$$= (n-1)\left(\frac{n-2}{2} + 1\right)$$

$$= \frac{n(n-1)}{2}$$



درجه دوم محدب

حالت میانگین: فرض کنید هر عنصری از ارایه با احتمال برابر بتواند عنصر محوری باشد. پس

$$A(n) = \sum_{p=1}^{n} \frac{1}{n}[A(p-1) + A(n-p)] + n - 1$$

$A_n = A_0 + A_{n-1} + n-1$

$\vdots$

$A_n = A_{n-1} + A_0 + n-1$

داریم

$$\sum_{p=1}^{n} [A(p-1) + A(n-p)] = 2\sum_{p=1}^{n} A(p-1).$$

پس

$$A(n) = \frac{2}{n}\sum_{p=1}^{n} A(p-1) + n - 1.$$

یا

$$nA(n) = 2\sum_{p=1}^{n} A(p-1) + n(n-1).$$

همچنین

$$(n-1)A(n-1) = 2\sum_{p=1}^{n-1} A(p-1) + (n-1)(n-2).$$

از دو معادله قبل داریم

$$nA(n) - (n-1)A(n-1) = 2A(n-1) + 2(n-1),$$

$$\frac{nA_n = (n+1)A_{n-1} + 2(n-1)}{n(n+1)}$$

یا

$$\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2(n-1)}{n(n+1)}.$$

حال اگر قرار دهیم $a_n = \dfrac{A(n)}{n+1}$, نتیجه می شود

$$< \frac{2}{n}$$

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)} \quad \text{for } n > 0 \qquad a_n \le a_{n-1} + \frac{2}{n}$$

$$a_0 = 0.$$

پس

$$A(n) = \theta(\mathrm{n} \log n)$$

بهترین حالت:

$$T(n) = \min_{0 \le q \le n-1} \left(T(q) + T(n-q-1) \right) + \Theta(n).$$

$T(n) \le \Theta(n \log n)$

آیا $T(n) \geq c\, n \log n$??

$\underbrace{T(q) + T(n-q-1)}_{*} \geq \underbrace{c\, q \log q + c(n-q-1)\log(n-q-1)}_{\min\limits_{0 \le q \le n-1} f(q)}$

$g(n) = n \log n$

$g'(n) = \log n + 1$

$g''(n) = \frac{1}{n} > 0$, $n > 0$

$f'(q) = c[\log q + 1 - \log(n-q-1) - 1]$

$= c[\log q - \log(n-q-1)] = 0$

$\log q = \log(n-q-1)$

$q = n-q-1$

$q = \frac{n-1}{2}$

$* \geq c(n-1)\log\frac{n-1}{2}$

$\Rightarrow T(n) \geq c(n-1)\log\frac{(n-1)}{2} + \Theta(n)$

$\overset{?}{\geq} c\, n \log n$

$(\Leftarrow)$ $c\, n \log(n-1) - c(n-1) - c\, n \log n + \Theta(n) \geq 0$

$c\, n \log\frac{n-1}{n} - c(n-1) + \Theta(n) \geq 0$

اگر c به اندازه کافی کوچک باشد
درست است

$T(n) = \Theta(n \log n)$

Having the worst case occur when they are sorted or almost sorted is *very bad*, since that is likely to be the case in certain applications.

To eliminate this problem, pick a better pivot:

1. Use the middle element of the subarray as pivot.
2. Use a *random* element of the array as the pivot.
3. Perhaps best of all, take the median of three elements (first, last, middle) as the pivot. Why should we use median instead of the mean?

- The main advantage of using the median-of-medians algorithm as the pivot selection method for quicksort is that **it guarantees a good balance of the partitions, regardless of the order of the original array**.

- Suppose that we want to sort an array $A[1..n]$ of length n.

  Quicksort picks a **pivot** element p uniformly at random.

- **Proposition**

  The expected number of comparisons made by randomized quicksort on an array of size n is at most $2n \ln n$.

Algorithm	Variation	Time complexity	Space complexity
Bubble sort	Best case	O(n)	O(1)
	Average case	O(n^2)	
	Worst case	O(n^2)	
Merge sort	Best case	O(n log n)	O(n log n) Can be improved to O(n)
	Average case		
	Worst case		
Quick sort	Best case	O(n log n)	O(log n)
	Average case	O(n log n)	O(log n)
	Worst case	O(n^2)	O(n)

- Together with its modest O(log n) space usage, **quicksort** is one of the most popular sorting algorithms and is available in many standard programming libraries.