

1 Summary on $\Theta(n^2)$ Sorting Algorithms (Section 7.2)

	Insertion	Bubble	Selection
Best-case	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Average-case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Worst-case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

2 Insertion Sort

Let $A[1 \dots n]$ be an array containing a sequence of n numbers to be sorted. The insertion sort algorithm is presented below

InsertionSort(A)	# of times the line executed
1: for j <- 2 to length[A]	$n-1+1 = n$
2: key <- A[j]	$n-1$
3: i <- j-1	$n-1$
4: while i>0 and A[i] > key	?
A[i+1] <- A[i]	?
i <- i-1	?
5: A[i+1] <- key	$n-1$

Let the cost be the number of times each statement is executed.

2.1 Best-case Complexity

Let t_j be the number of times the while loop in Line 4 is executed for that value of j , for each $j = 2, 3, \dots, n$, where $n = \text{length}[A]$, times of executions for line 4 is

$$\sum_{j=2}^n t_j = t_2 + t_3 + \dots + t_n.$$

If Line 4 is executed t_j times, then each of Line 5 and Line 6 will be executed $(t_j - 1)$ times. That is, times of executions for Line 5 is $\sum_{j=2}^n (t_j - 1)$, and for Line 6 is also $\sum_{j=2}^n (t_j - 1)$. For the best case, we know that the array is already sorted. For each $j = 2, 3, \dots, n$, we then find that $A[i] \leq \text{key}$ in Line 4 when i has its initial value of $j - 1$. Thus, $t_j = 1$ for $j = 2, 3, \dots, n$ and the best-case of the times of executions is

$$\begin{aligned}
 T(n) &= 3(n-1) + n + \sum_{j=2}^n t_j + 2 \sum_{j=2}^n (t_j - 1) \quad \left(\text{note that } \sum_{j=2}^n (t_j - 1) = 0 \right) \\
 &= 4n - 3 + \sum_{j=2}^n 1 \\
 &= 4n - 3 + n - 1 \\
 &= 5n - 4.
 \end{aligned}$$

Thus $T(n)$ is a linear function, i.e., $T(n) \in O(n)$.

2.2 Worst-case Complexity

For worst-case, the array is in reversely sorted order (decreasing order). We must compare each element $A[j]$ with each element in the entirely sorted subarray $A[1 \dots j-1]$, and so $t_j = j$ for $j = 2, 3, \dots, n$. Hence, the worst-case running time is

$$\begin{aligned}
 T(n) &= 3(n-1) + n + \sum_{j=2}^n t_j + 2 \sum_{j=2}^n (t_j - 1) \\
 &= 4n - 3 + \sum_{j=2}^n j + 2 \sum_{j=2}^n (j-1) \\
 &= 4n - 3 + \frac{n(n+1)}{2} - 1 + 2\left(\frac{(n-1)n}{2}\right) \\
 &= 4n - 3 + \frac{n^2}{2} + \frac{n}{2} - 1 + n^2 - n \\
 &= \frac{3n^2}{2} + \frac{7n}{2} - 4
 \end{aligned}$$

Hence, in worst-case, $T(n) \in O(n^2)$.

3 QuickSort (Section 7.5)

We first review what is lower bounds and upper bound. Remember that worst-case running time $T(n) = \max\{t(x) : x \text{ has size } n\}$. An upper bound is a function $g(n)$ s.t. $f(n) \in O(g(n))$ (essentially, $T(n) \leq g(n)$). A lower bound on $T(n)$ is a function $f(n)$ s.t. $T(n) \in \Omega(f(n))$ (essentially, $T(n) \geq f(n)$). Generally, to prove that $g(n)$ is an upper bound on $T(n)$, we argue that for all x of size n , $t(x) \leq g(n)$. Generally, to prove that $f(n)$ is a lower bound on $T(n)$, we exhibit one specific input x , for which we can show $t(x) \geq f(n)$.

The average-case running time of an algorithm will always be less than or equal to the worst-case by definition. we have already shown an example (ListSearch in Lecture Two) where the two are asymptotically the same. As we will see below, that they can also be asymptotically different, with the average-case becoming much better.

QuickSort(S) (S is the list to sort)

```

1: if |S| <=1 return S
2: select pivot p in S
3: partition elements of S into:
    L = elements of S less than p
    E = elements of S equal to p
    U = elements of S greater than p
4: return QuickSort(L); E; QuickSort(U)

```

Standard QuickSort: we select the pivot by simply letting p be the first element in S .

Example 1. $S = [15, 25, 12, 2, 37]$.

$p = 15$, $L = [12, 2]$; $M = [15]$; $U = [25, 37]$

$p = 12$, $L = [2]$; $M = [12]$; $U = []$; $p = 25$, $L = []$; $M = [25]$; $U = [37]$;

2, 12, 15, 25, 37.

3.1 Worst-case Complexity

How many comparisons are performed in the worst-case on a list of size n ?

Upper bound: We show $T(n) \in O(n^2)$. All of the comparisons occur between the pivot and other elements in the list. Each element will be compared at most once with every other element. Hence, there can be no more comparisons than there are pairs of elements in the list, i.e.,

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

Lower bound: We show $T(n) \in \Omega(n^2)$. Consider the input $S = [n, n-1, n-2, \dots, 2, 1]$. 1st call: $n-1$ comparisons with pivot “n”;

2nd call: $n-2$ comparisons with pivot “n-1”;

3rd call: $n-3$ comparisons with pivot “n-2”;

... ..;

$(n-1)^{st}$ call: 1 comparisons with pivot “2”.

Total is $(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$.

Conclusion: $T(n) \in \Theta(n^2)$, more precisely, $T(n) = \frac{n(n-1)}{2}$.

Where is the “quickness” in QuickSort?

3.2 Average-case Complexity of QuickSort (Section 14.2.4)

First we define the sample space S_n of all inputs of size n :

$$S_n = \{\text{all permutation of } [1, 2, 3, \dots, n]\},$$

assuming a uniform probability distribution on S_n .

Next let $t_n(S)$ be the number of comparisons performed by QuickSort on input $S \in S_n$. We want to compute $T'(n) = E(t_n(S); S \in S_n]$.

Observation:

1. In the first call, each $i \in \{1, 2, \dots, n\}$; is equally likely to be the pivot.
2. After the partition, L is a random permutation of $[1, 2, \dots, i-1]$ and u is a random permutation of $[i+1, i+2, \dots, n]$.

This allow us to write: if the pivot is i , then

$$T'(n) = n-1 + T'(n-1) + T'(n-i)$$

$$T'(n) = n-1 + \frac{2}{n} \sum_{j=0}^{n-1} T'(j) (*)$$

$$T'(n-1) = n-2 + \frac{2}{n-1} \sum_{j=0}^{n-2} T'(j)(**)$$

$$nT'(n) = n(n-1) + n \frac{2}{n} \sum_{j=0}^{n-1} T'(j) \text{ (from *)}$$

$$(n-1)T'(n-1) = (n-1)(n-2) + (n-1) \frac{2}{n-1} \sum_{j=0}^{n-2} T'(j) \text{ (from **)}$$

$$nT'(n) - (n-1)T'(n-1) = 2(n-1) + 2T'(n-1),$$

$$nT'(n) = (n+1)T'(n-1) + 2(n-1),$$

Divide both sides by $n(n+1)$

$$\frac{T'(n)}{n+1} = \frac{T'(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

let $A(n) = \frac{T'(n)}{n+1}$, we have

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

and

$$A(0) = \frac{T'(0)}{1} = 0.$$

hence,

$$\begin{aligned} A(n) &= \sum_{i=1}^n \frac{2(i-1)}{i(i+1)} = \sum_{i=1}^n \frac{2i}{i(i+1)} - \sum_{i=1}^n \frac{2}{i(i+1)} \\ &= 2 \sum_{i=1}^n \frac{1}{(i+1)} - 2 \sum_{i=1}^n \left(\frac{1}{i} - \frac{1}{i+1} \right) \\ &= 2\Theta(\log n) - 2\left(1 - \frac{1}{n+1}\right) = \Theta(\log n). \end{aligned}$$

Hence,

$$T'(n) = (n-1)A(n) = \Theta(n \log n).$$

Note that, The sum of reciprocals from 1 to n , is called the Harmonic Series and written as \mathcal{H}_n , has a value of $\log_e n + O(1)$.

3.3 Randomized QuickSort

How can we make QuickSort quick for all inputs?

- Average-case analysis works only if each permutation equally likely. In practice, this may not be true.
- Instead of relying on unknown distribution of inputs, randomize algorithm by picking random element as pivot. This way, random behaviour of algorithm on any fixed input is equivalent to fixed behaviour of algorithm on a uniformly random input, i.e., expected worst-case time of randomized algorithm *on every single input* is $\Theta(n \log n)$.
- In general, randomized algorithms are good when there are many good choices but it is difficult to find one choice that is guaranteed to be good.