Algorithms Design and Analysis [ETCS-301]

Dr. A K Yadav
Amity School of Engineering and Technology
(affiliated to GGSIPU, Delhi)
akyadav1@amity.edu
akyadav@akyadav.in
www.akyadav.in
+91 9911375598

August 20, 2019

Complexity analysis I

Analyzing an algorithm means predicting the resources that the algorithm requires. Resources may be memory, communication bandwidth, computer hardware or CPU time. Our primary concern is to measures the computational time required for the algorithm.
Running time:-The running time of an algorithm is the number of primitive operations or steps executed on a particular input.
Why do we normally concentrate on finding only the worst-case running time?

1. The worst-case running time of an algorithm gives us an upper bound on the running time for any input. So it guarantees that the algorithm will never slower than this.
2. In real applications, worst case normally occurs for example searching a non existing data.

Complexity analysis II

3. Best case is like an ideal case which guarantees that the algorithm will never faster than stated. Based upon this we can't allocate the resources.
4. Average case normally perform as worst case because normally we take average case as average of best and worst or best for half size input and worst for other half size.

Complexity analysis: Insertion sort I

Insertion-Sort(A,N)	Cost	Times
1. for $j = 2$ to N	c_1	n
2. $key = A[j]$	c_2	n-1
Insert $A[j]$ in sorted $A[1]$ to $A[j-1]$		
3. $i = j - 1$	c_3	n-1
4. while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
5. $A[i+1] = A[i]$	c_5	$\sum_{j=2}^{n}(t_j - 1)$
6. $i = i - 1$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
while-end		
7. $A[i+1] = key$	c_7	n-1
for-end		

Complexity analysis: Insertion sort II

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

$$\Rightarrow T(n) = an + b + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1)$$

Now consider different cases:

Complexity analysis: Insertion sort III

1. **Best Case:** The algorithm performs best if $key \leq A[i]$ for every value of j in step 4.
   Then it executes only once for each value of j and total of n-1 times.
   Step 5 and 6 will not be execute at all.
   This is the case when array is already sorted

   $$T(n) = an + b = O(n)$$

2. **Worst Case:** The algorithm performs worst if $key > A[i]$ for each value of j and stops only when $i < 1$ in step 4.
   Then it will execute always j times for each value of $j = 2, 3, \ldots, n$

Complexity analysis: Insertion sort IV

so

$$\sum_{j=2}^{n} j = \frac{(n-1)(2+n)}{2}$$

and step 5 and 6 will execute

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

This is the case when array is already sorted in reverse order

$$T(n) = an^2 + bn + c = O(n^2)$$

Complexity analysis: Merge Sort I

MERGE-SORT(A, p, r)

1 if $p < r$
    2 $q = (p + r)/2$
    3 MERGE-SORT(A, p, q)
    4 MERGE-SORT(A, q+1, r)
    5 MERGE(A,p,q,r)

MERGE(A, p, q, r)

6 $n_1 = q - p + 1$

7 $n_2 = r - q$

8 Let $L[1..n_1]$ and $R[1..n_2]$ be new arrays

9 for $i = 1$ to n_1
    10 $L[i] = A[p + i - 1]$

11 for $j = 1$ to n_2
    12 $R[j] = A[q + j]$

13 $i = 1$

Complexity analysis: Merge Sort II

14 $j = 1$

15 for $k = p$ to r

  16 if $L[i] \leq R[j]$

    17 $A[k] = L[i]$

    18 $i = i + 1$

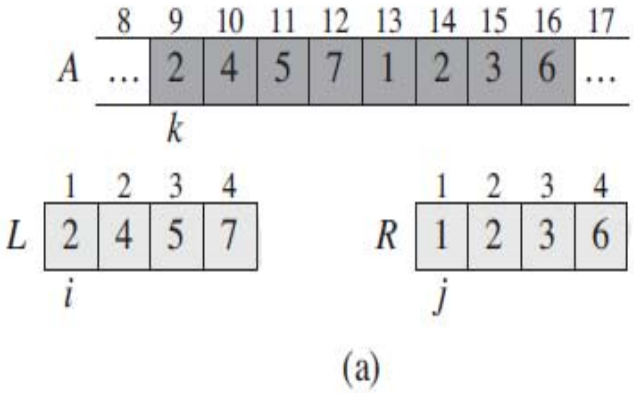
  19 else

    20 $A[k] = R[j]$

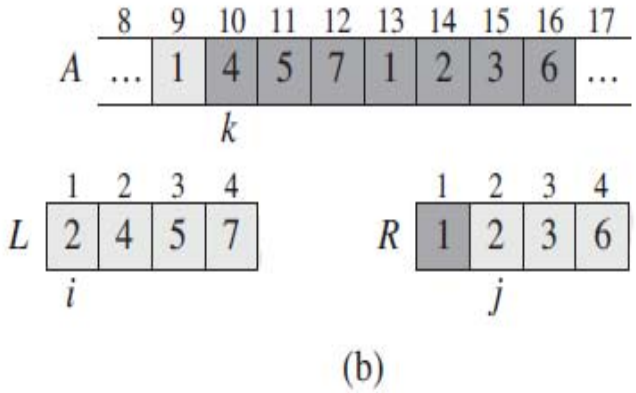
    21 $j = j + 1$

Complexity analysis: Merge Sort III

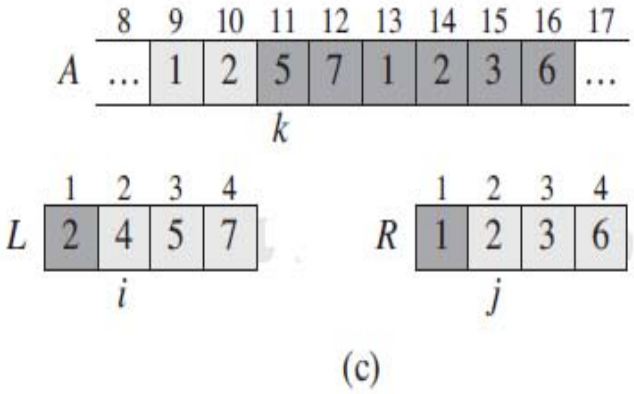
Working of Merge



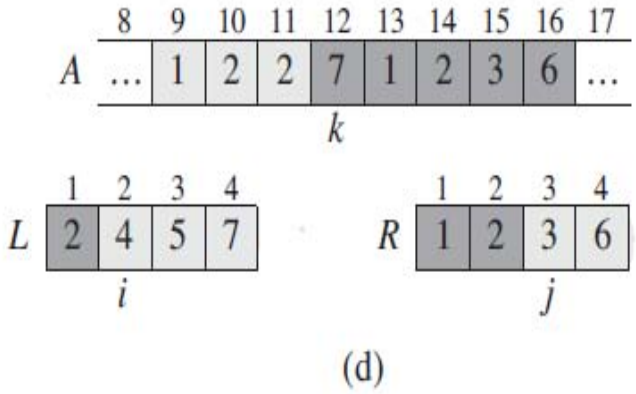
(a)



(b)



(c)



(d)

Analysis:

Complexity analysis: Merge Sort IV

- First call of merge-sort will be MERGE-SORT(A, 1, n) for input size n.
- q is half of n,$q = \frac{n}{2}$ in step 2.
- Every call of merge-sort divides the size in half and double the sub-problems.
- So there will be only lg n calls of merge-sort and sum of size of all sub-problems is n
- There is only one call for each merge-sort call
- So total calls of the merge will be lg n
- Step 6 to 8 will be executed lg n times
- Step 9 will be executed n_1 lg n times
- Step 10 will be executed n_1 lg n times
- Step 11 will be executed n_2 lg n times
- Step 12 will be executed n_2 lg n times

Complexity analysis: Merge Sort V

- Total of Step 9 to 12 will be executed $2(n_1 + n_2) \lg n$ times
- Step 13 & 14 will be executed $\lg n$ times
- For each call of $\lg n$ step 15 will be executed $n \lg n$ times
- Step 16 will be executed $n \lg n$ times
- Every time either step 17 & 18 or step 20 & 21 will be executed and sum of these will be $n \lg n$.
- After adding cost of all these step
  $T(n) = an \lg n + bn + c = O(n \lg n)$

Thank you

Please send your feedback or any queries to akyadav1@amity.edu, akyadav@akyadav.in or contact me on +91 9911375598