Date ..../..../.....

Algorithm Design and Analysis

Assignment - 1

Mandeep Singh
I-123
00814807720

① What are different Asymptotic Notations? Explain with Examples.

=> • Asymptotic analysis is used to evaluate the performance of an algorithm in terms of input size.

• The basic idea of asymptotic analysis is to measure the efficiency of algorithm that doesn't depends on the machine specific constants.

• The mathematical tools to represent the time complexity of algorithm for asymptotic analysis are called as asymptotic notations.

* There are 3 notations to measure the time complexity of a program namely :-

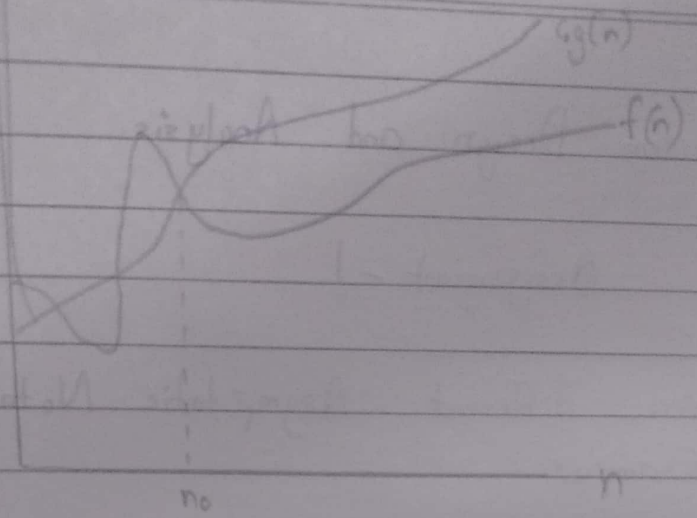
Big - O, Big - Ω and Big - Θ.

a) Big - O :-

• The notation defines an upper bound of an algorithm.

Special

Date ..../..../.....

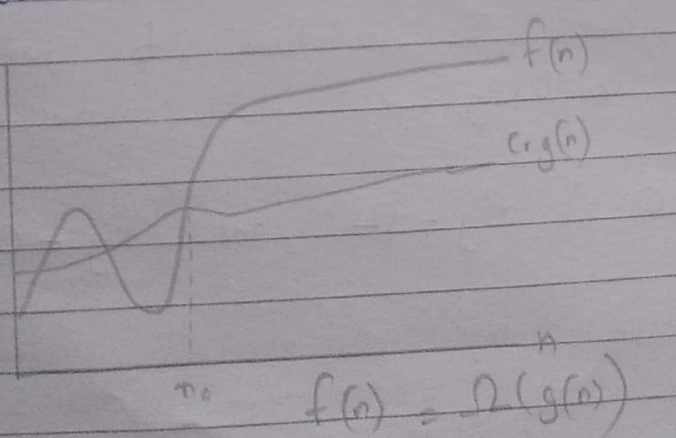


$f(n) = O(g(n))$

- The function $f(n) = O(g(n))$ if and only if $f_n <= c.g(n)$, for all $n >= n_0$ where c and n_0 are constants.
- Here, $g(n)$ is known as upper bound of on values of $f(n)$.
- Eg. $f(n) = 3n + 3$, $g(n) = 4n$.

b) Big-Ω :-

- Ω notation provides an asymptotic lower bound.
-



$f(n) = \Omega(g(n))$

- The function $f(n) = \Omega(g(n))$ if $f(n) >= c.g(n)$

Spiral

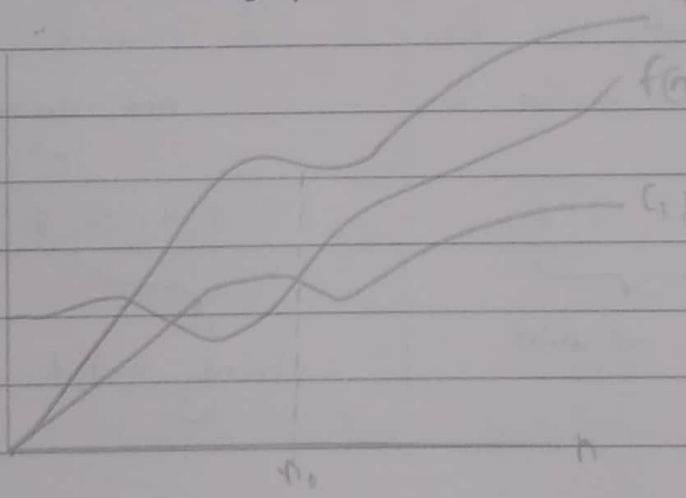
Date ..../..../.....

for all $n >= n_0$, where c and n_0 are constants.

- Here, $g(n)$ is known as lower bound value of $f(n)$.
- E.g. $f(n) = 3n + 2$ and $g(n) = 3n$.

c) Big - Θ :-

- The theta notation bounds a function from above and below, so it defines exact asymptotic behaviour. (tighty bounds)



$f(n) = \Theta(g(n))$

- The function $f(n) = \theta(g(n))$ if $c_1 g(n) <= f(n) <= c_2 g(n)$ for all $n >= n_0$, where c_1, c_2 and n_0 are constants.
- Eg. $f(n) = 3n + 2$, $g(n) = n$, $c_1 = 3$ & $c_2 = 4$.

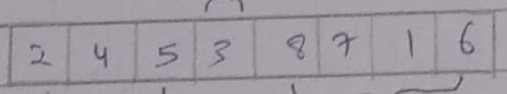
Date ..../..../.....

(2). Write and observe complexity analysis of Insertion Sort, Merge Sort, Radix Sort and Quick Sort.

⇒ a) Insertion Sort :-

(1)

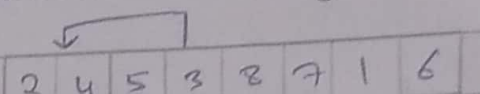
current item



already sorted unsorted

(2)

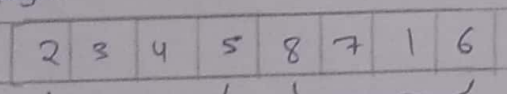
insert where it belongs



already sorted unsorted

(3)

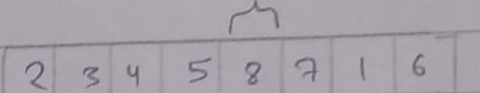
shift other elements over and already sorted section is larger.



already sorted not sorted

(4)

new current item



already sorted unsorted

- idea : at step 'k', put the k^{th} element in correct position among the first k elements.

Ex.
```
for ( int i=0 ; i<n ; i++) {
    // find index to insert into
    int newIndex = findPlace (i);
    // insert and shift nodes over
    shift (newIndex, i);
}
```

- when bop index is i, first i elements are sorted from the first i elements in array.

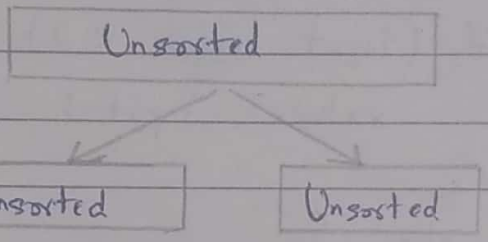
Scanned with CamScanner

Date ..../..../.....

- Runtime : Best $\Omega(n)$ Worst $O(n^2)$ Average $O(n^2)$

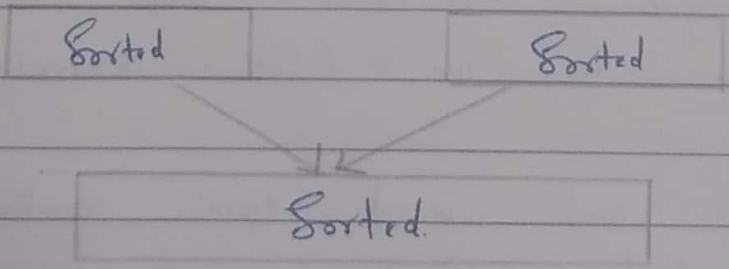
b) Merge Sort :-

- Divide : split array roughly into half



- Conquer : Return away when length ≤ 1

- Combine : Combine two sorted arrays using merge



Date ..../..../.....

Merge sort : Pseudocode

→ Core idea : split array in half, sort each half, merge back together. if the array has size 0 and 1, just return it unchanged

```
mergeSort(input) {
    if (input.length < 2) {
        return input;
    }
    else {
        smallerHalf = sort(input[0, ---, mid]);
        largerHalf = sort(input[mid+1, ---]);
        return merge(smallerHalf, largerHalf);
    }
}
```

→ Runtime :-

→ Subdivide the array in half each time : $O(\lg(n))$ recursive calls.

→ merge is an $O(n)$ traversal of each level.

So, the best and worst case runtime is the same : $O(n.\lg(n))$.

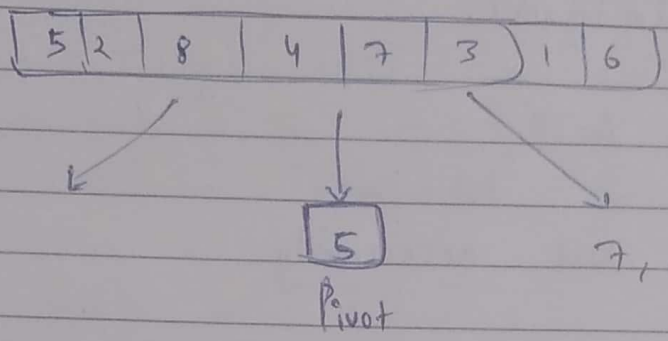
→ Stable → Yes !

→ Inplace → No.

Date ..../..../.....

6) Quick sort.

Divide : Split array around a pivot.



Conquer :- Return array when length ≤ 1

Combine :- Combine sorted partition and pivot.

⇒ Analysis :-

- Best-case : Pivot is always the median

$T(0) = T(1) = 1$

$T(n) = 2T(n/2) + n$ (linear time partition)

Same recurrence as merge sort: $O(n \cdot \log(n))$

- Worst-case : Pivot is always smallest or largest element.

$T(0) = T(1) = 1$

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

Basically same recurrence as selection sort : $O(n^2)$

Date ..../..../.....

- Average - case (e.g., with random pivot)

$$T(n) = n + \frac{(n-1)!}{n!}\left[\sum_{i=1}^{n} T(i-1) + T(n-i)\right]$$

$O(n \log(n))$.

- Inplace : Yes !
- Stable : Not necessarily.

d) Radix Sort

- Radix = "the base of a number system".
  - Bucket sort on one digit at a time
    - no. of bucket = radix
    - starting with least significant digit
    - keeping sort stable.
  - Do one pass per digit.
  - Invariant : after k passes (digits), the last k digits are sorted.
- Analysis : Input = n / No. of buckets = B / No. of passes = P

work per pass is 1 bucket sort : $O(B+n)$
Total work is $O(P(B+n))$

Compared to comparison sorts, sometime a win, but often not. String of English letter up to 15.

Spiral

Date ..../..../.....

(3) Master Method :-

The master method is used for solving the following type of recurrence :-

$T(n) = a\,T(n/b) + f(n)$ with $a \geq 1$ and $b \geq 1$ be constant & $f(n)$ be a function and $\frac{n}{b}$ can be interpreted as,

Let $T(n)$ is defined on non-negative integers by the recurrence.

$$T(n) = a \cdot T(n/b) + f(n)$$

In the function to the analysis of a recursive algorithm, the constants and function take on the following significance :-

- → n is the size of the problem.
- → a is the no. of subproblem in the recursion.
- → n/b is the size of each subproblem.
- → $f(n)$ is the sum of the work done outside the recursive calls.
- → It is not possible always bound the function according to the requirement, so we make 3 cases :-

- Case 1 :- If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then it follows :

$$T(n) = \Theta(n^{\log_b a})$$

Spiral

Date ..../..../.....

Ex:-

$T(n) = 8T(\frac{n}{2}) + 1000n^2$ apply master on it

Sol:-

Compare $T(n) = 8T(n/2) + 1000n^2$ with

$T(n) = aT(n/b) + f(n)$ with $a \geq 1$ & $b \geq 1$

put all values, $f(n) = O(n^{\log_b a - \epsilon})$

$1000n^2 = O(n^{3-\epsilon})$

put $\epsilon = 1$, we get $= O(n^2)$

$T(n) = \Theta(n^{\log_b a})$

Therefore, $T(n) = \Theta(n^3)$

- Case 2 : If it is true, for some constant $k \geq 0$ that,

$f(n) = O(n^{\log_b a} \log^k n)$ then it follows that :

$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

Ex:-

$T(n) = 2T(n/2) + 10n$, solve the recurrence by using the master method.

As compare the given problem with $T(n) = aT(n/b) + f(n)$ with $a \geq 1$ and $b \geq 1$ $a = 2$,

put all values in $f(n) = O(n^{\log_b a} \log^k n)$, we will get

Spiral

Date ..../..../.....

$\log n = \theta(n^1) = \theta(n)$ which is true.

Therefore : $T(n) = \theta(n^{\log_b a} \log^{k+1} n)$

$= \theta(n \log n)$

- Case 3 : If it is true $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and it also true that : $a\, f(n/b) \leq c f(n)$ for some constant $c < 1$ for large value of n, then :

$$T(n) = \theta(f(n))$$

Example : $T(n) = 2\, T(n/2) + n^2$

Sol. : Compare the given problem with $T(n) = a.T(n/b) + f(n)$ with $a \geq 1$ & $b > 1$

put all values,

$n^2 = \Omega(n^{1+\epsilon})$ put $\epsilon = 1$

$n^2 = \Omega(n^{1+1}) = \Omega(n^2)$

Now we will also check the second condition :

$2(n/2)^2 \leq cn^2 \Rightarrow \frac{1}{2} n^2 \leq cn^2$

$T(n) = \theta(n^2)$