1) Student Name : Venkatesh G.P

2) USN : 2GI19CS175

3) BE : BE

4) Semester : 4

5) Course Name : Design Analysis of Algorithm

6) Course Code :

7) Name of College : KLS GIT

8) Date & Time : 04-06-2021 ; 3:00 PM

9) Mobile No : 9972287036

10) I hereby declare that the above mentioned information is true to best of my knowledge

11) Signature

5.) Binary Search Algorithm

→ function : ~~Binary_Search~~ (A, n, T)
Given an array A of n elements & target value T.

Step 1: Set L to 0 & R to n-1
Step 2: If L > R, the search terminates as unsuccessful
Step 3: Set m (position of middle element) to floor of (L+R)/2
Step 4: If $A_m < T$, set L to m+1 & goto step 2
Step 5: If $A_m > T$, set R to m-1 & goto step 2
Step 6: Now $A_m = T$, the search is done, return m

→ Efficiency of algorithm

* Best Case: A best case would be if the search element is in middle of the entire list, which would lead to O(1) time complexity.

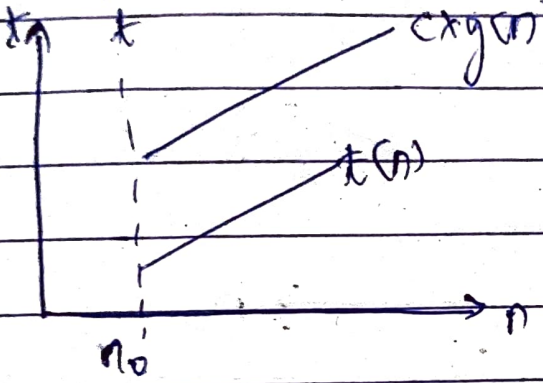
* Worst Case: In worst case, binary search runs in logarithmic time, making O(log N) comparisons which usually occurs if search element is in the corner ends of the array.

3.)

→ Big Oh (O): A function $f(n)$ is said to be in $g(n)$, denoted by $t(n) \in O(g(n))$ if $t(n)$ is bounded above by some constant multiple $g(n)$ for all large n i.e if there exists some positive constant c, & some non-negative integer n_0 such that $t(n) \leq c \cdot g(n)$ for all $n > n_0$

* Eg: The function $2n+3 \leq 6n$, $2n+3$ is under big oh of n for $c = 6$ & $n_0 \geq 1$

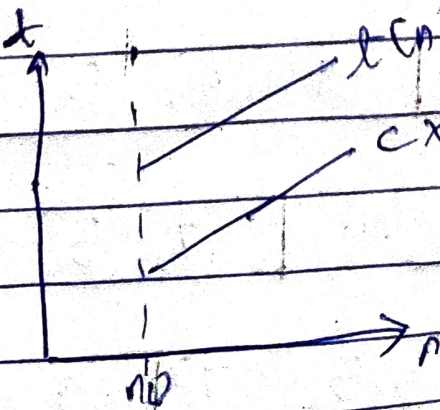
* Graphically, big-oh can be expressed as



→ Big Omega (Ω): A function $t(n)$ is said to be in Ω of $g(n)$ denoted by $t(n) \in \Omega g(n)$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$, for all large n, i.e if there exists some positive const c & some non-negative integer n_0 such that $t(n) \geq cg(n)$ for all $n \geq n_0$

* Eg: $2n+3 \in \Omega(n)$, for $c = 1$ & $n_0 \geq 1$

* Graph:



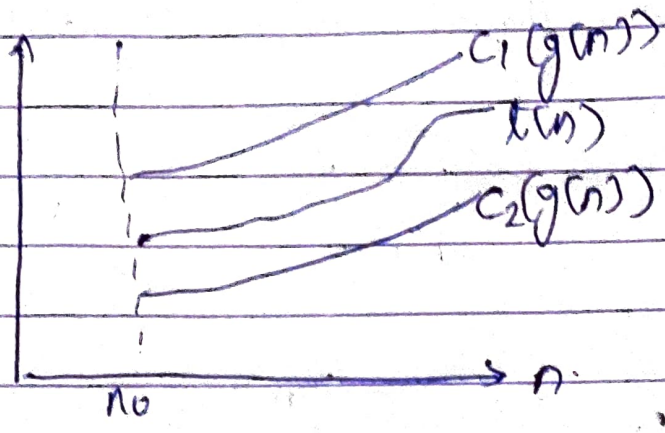
3.) → Big Theta (Θ): A function $t(n)$ is said to be under $\Theta(n)$ & is denoted by $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded above & below by constant multiples of $g(n)$ for all large n. Such that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \quad \forall \; n \geq n_0$$

* Eg: $t(n) = 2n+3 \in \Theta(n)$, as it is bounded above & below by const 1 & 5.

$1 * n \leq 2n+3 \leq 5 * n$, $n_0 = 1$.

* Graph:



6.) Given A = [4, 2, 7, 9, 1, 5, 10, 8]

Applying Selection sort

A	4	2	7	9	(1)	5	10	8
A	1	(2)	7	9	4	5	10	8
A	1	2	7	9	(4)	5	10	8
A	1	2	4	9	7	(5)	10	8
A	1	2	4	5	(7)	9	10	8

6.)

A	1	2	4	5	7	9	10	(8)
A	1	2	4	5	7	8	10	(9)
A	1	2	4	5	7	8	9	10

∴ The sorted list will be: A = [1, 2, 4, 5, 7, 8, 9, 10]

* Efficiency: The time efficiency of selection sort is quadratic.
  1.) Best case: $O(n^2)$ Time complexity.
  2.) Worst Case: $O(n^2)$ Time complexity.

7.) → General Plan for Non recursive algorithms:

* Decide a parameter N indicating input size.
* Determine ~~worst case~~ algorithmic basic operation: Linear search in the given question
* Determine best, worst & average case of input size N.
  Best case for Linear search:
  → Best case: O(1), element found at index 0
  → Average case: O(N)
  → Worst case: O(N), element found at index N.
* Set up a sum for the number of times the basic operation is executed.

→ Algorithm for Linear Search:

for i ← 0 to n do

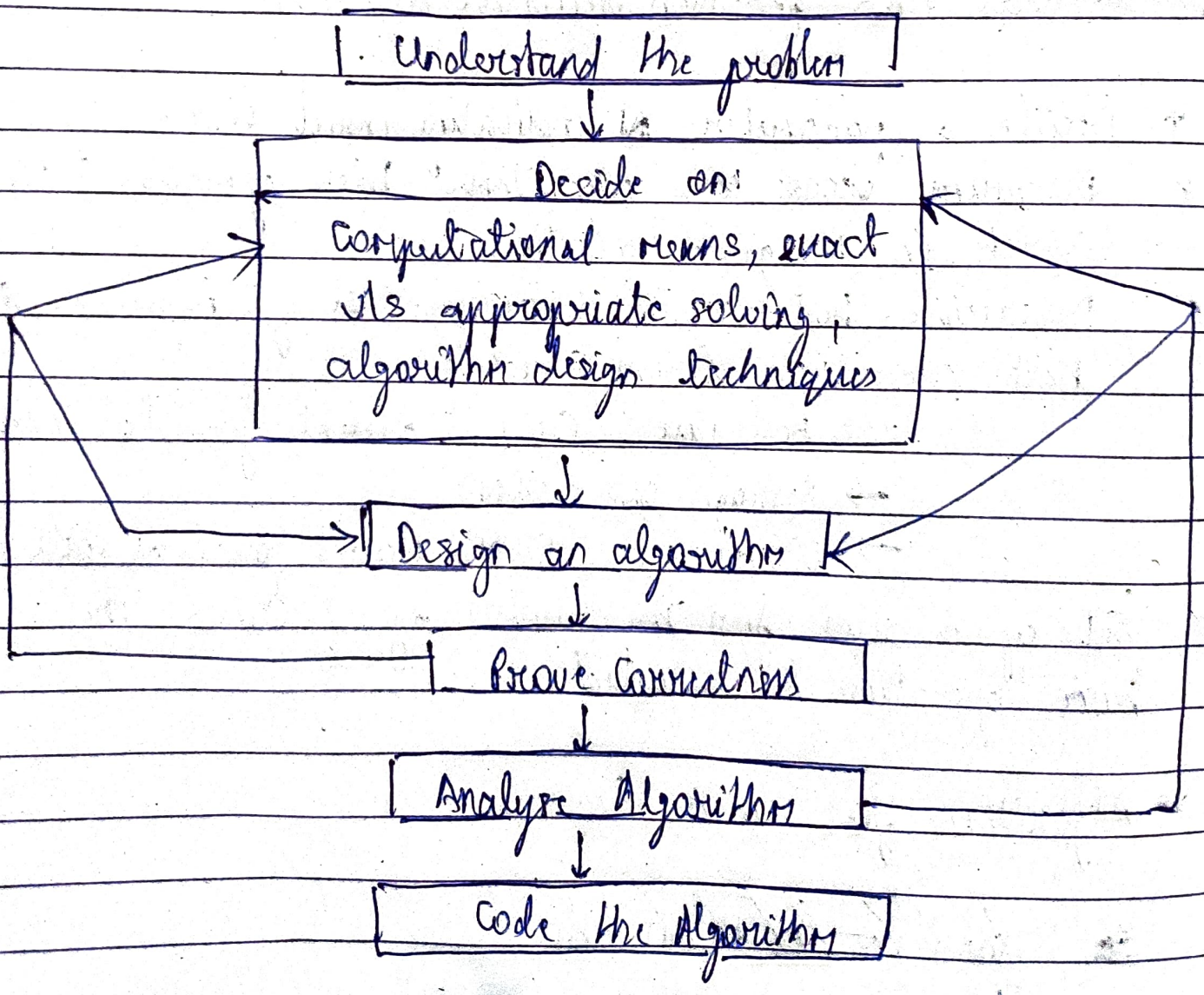
7.)
```
for i ← 0 to n-1 do
    if A[i] == T
        return i
```

where, A is the array, n is array size & T is the target element.

1.)

→ Space Efficiency: There are

2.) Algorithm approach can be used/applied to solve any real world problem



2)

* 1) Understand the problem: A algorithm designer should interpret the problem definition & understand requirements of problem.

2) Decide upon:
   * The capabilities of computational device for which algorithm needs to be designed.
   * Choose appropriate data structure

3) Design an algorithm:
   In this step the actual solution of problem is detected.

4) Proving correctness: It is important to prove an check the correctness of the output.

5) Analyse the algorithm: The correctly proven solution from previous steps have to be compared an analyzed with respect to some parameters to arrive to an optimum solution.

6) Code the algorithm: Convert the given algorithm into an executing program written in some programming language of choice.

* Analysis of Algorithm: After designing solution for a given problem, its important to study an analyse the solution to arrive at the best solution from the available solutions. The different ways of analysis of algorithms are time efficiency, space efficiency, correctness & optimality.