ALGORITHM ANALYSIS EXAM

1) Solve the problem of finding the distance between the two FURTHEST (according to digital values) numbers in the array below with 3 different methods and compare their efficiencies.

Method_1: Simple Linear Search

Initialize values of min and max as minimum and maximum of the first two elements respectively. Starting from 3rd, compare each element with max and min, and change max and min accordingly (i.e., if the element is smaller than min then change min, else if the element is greater than max then change max, else ignore the element)

After find min and max value, get their indexes and subtract them.

Distance = indexOf(maxValue) – indexOf(minValue)

Time Complexity: O(n)

In this method, the total number of comparisons is 1 + 2(n-2) in the worst case and 1 + n – 2 in the best case.
In the above implementation, the worst case occurs when elements are sorted in descending order and the best case occurs when elements are sorted in ascending order.

Method_2: Tournament Method

Divide the array into two parts and compare the maximums and minimums of the two parts to get the maximum and the minimum of the whole array

```
Pair MaxMin(array, array_size)
   if array_size = 1
      return element as both max and min
   else if arry_size = 2
      one comparison to determine max and min
      return that pair
   else    /* array_size  > 2 */
      recur for max and min of left half
```

```
    recur for max and min of right half
    one comparison determines true max of the two candidates
    one comparison determines true min of the two candidates
    return the pair of max and min
```

Distance = indexOf(maxValue) – indexOf(minValue)

Time Complexity: O(n)

Method_3: Compare in Pairs

If *n* is odd then initialize *min* and *max* as first element.
If *n* is even then initialize *min* and *max* as minimum and maximum of the first two elements respectively.
For rest of the elements, pick them in pairs and compare their
maximum and minimum with *max* and *min* respectively.

Distance = indexOf(maxValue) – indexOf(minValue)

Time Complexity: O(n)

Compare Efficiencies

Second and third approaches make the equal number of comparisons when n is a power of 2.
In general, method 3 seems to be the best.

2) What is the “worst case analysis” of the function f(n) = 100n + 5 for different values of n?

For this problem we have to use Big O notion.

|f(n) | <= c * |g(x)| if and only if there is a c and an k such that for all n :

n is bigger than k and ;

- |100n + 5| <= 100n + 5n
- |100n + 5| <= 105|n|

Here we assume k = 1 for n > k and we get c = 105

3) Explain the concepts below.

a) **Np Problem** : A problem is assigned to the NP (nondeterministic polynomial time) class if it is solvable in polynomial time by a nondeterministic Turing machine.A P-problem (whose solution time is bounded by a polynomial) is always also NP. If a problem is known to be NP, and a solution to the problem is somehow known, then demonstrating the correctness of the solution can always be reduced to a single P (polynomial time) verification. If P and NP are not equivalent, then the solution of NP-problems requires (in the worst case) an exhaustive search.Linear programming, long known to be NP and thought not to be P, was shown to be P by L. Khachian in 1979. It is an important unsolved problem to determine if all apparently NP problems are actually P.A problem is said to be NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problem. It is much easier to show that a problem is NP than to show that it is NP-hard. A problem which is both NP and NP-hard is called an NP-complete problem.

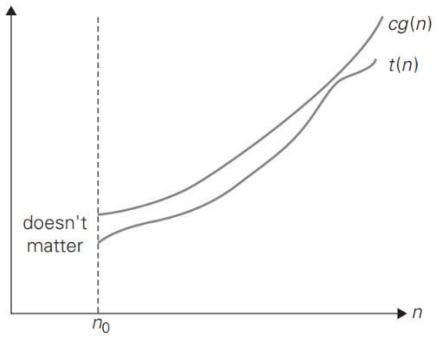
b) **Algorithm Analysis :** Analysis of computer algorithms; It deals with basic questions about how the algorithm works, how long it takes to work, whether it ends, whether an algorithm can be written on any subject. This analysis begins by estimating the running time of an algorithm. For example, all steps, all stages of the sales process to be carried out on a shopping site, to whom the product will be sold primarily, bank, payment and after-sales transactions can be carried out using many different algorithms. Algorithm analysis; focuses on the problems on the existing algorithm, which has been designed, offers methods to solve the problems of using which algorithm to ensure that the system operates faster and with high performance.

c) **Approximate Solution :** Approximating Solutions, also called Trial and Error, or Trial and Improvement, is used for calculating values when an equation cannot be solved using another method.The process involves estimating a start value, deriving the answer from the equation, and then improving the next estimate. This process is repeated until the required accuracy is achieved.The working is normally set out in a table with three columns: the value being tested; the calculation with that value; and a comment about the result

d) **Divide and Conquer :** Divide and conquer is an important algorithm design technique in computer science. It is based on recursively solving a problem by breaking it down into two or more similar sub-problems. Then, a solution is chosen among the solutions of these sub-problems. Many algorithms use this method, including search (Mergesort, Quicksort), "Discrete Fourier Transforms" (FFTs). D&C algorithms are performed as recursive procedures. You can also use it in a non-recursive way where the results are stored in a data type.

e) **Average Case :** The efficiency of an algorithm "A" is measured by the amount of computational resources used, in the first place time (number of computation steps) and space (amount of memory cells). These values may depend on the individual inputs given to "A". Thus, in general it is infeasible to give a complete description of the efficiency of an algorithm, simply because the amount of data grows exponentially with respect to the size of the inputs.

4) Give 3 statements about big-oh notation and prove these statements by using the graphic below.



Big-oh notation: $t(n) \in O(g(n))$.

Statement 1: a function t(n) is said to be in O(g(n)) , denoted t(n) ∈ O(g(n))

Statement 2: İf t(n) is bounded above by some constant multiple of g(n) for all large n, if There exist some positive constant c and some nonnegative integer n_0 .

Statement 3: we can see t(n) is always below of cg(n) so we can say that ;

t(n) <= cg(n) for all n >= n_0