# CSE 221 Theory Assignment 01 & 02 [HFN Section: 09 & 26]

## Part 1: Iterative Time Complexity

**Q1. Find the time complexity of the following codes,**

**a.**

```
c = 0
x=30
while c<n:
x= x + c
c += 2
```

**b.**

```
x = 0
i = 1
while(x<n):
x = x + 2*i
i+=1
```

**c.**

```
x=100
for h in range(x*x):
print(“ :( ”)
```

**d.**

```
x = 0
i = 1
while(x<n):
x = x + i
i+=2
```

**e.**

```
int p = 0;
for (i = n / 2; i > 1; i /= 6) {
     for (j = 2; j <= n; j *= 4) {
          for (k = 0; k <= j; k *= 3) {
               p = p + n / 2;
          }
     }
}
```

**f.**

```
int p = 0;
for (i = n; i > 1; i -= 1) {
     for (j = 2; j <= n; j += 1) {
          p = p + n;
          if (p > (n ^ i)) {
               break;
          }} }
```

```
    for (i = n / 2; i > 1; i /= 6) {
        for (j = 2; j <= i; j *= 4) {
            for (k = 0; k <= j; k *= 3) {
                p = p + n / 2;
            }
        }
    }
```

g. Explain why the statement, "The running time of algorithm A is at least O($n^2$)" is meaningless.

**Part 2: Search**

[you can write pseudocode/ programmable code/ step by step instructions if you want but NOT EXPLANATION PASSAGES, ONLY LOGICAL INSTRUCTIONS
]

Q1. You are given an array containing N distinct integers in a wave-like sequence. Meaning, the numbers in the beginning are in ascending order, and after a specific position, they are in descending order. For example: [1, 3, 4, 5, 9, 6, 2, -1]

a. You have to find the maximum number of this sequence. Can you devise an efficient algorithm such that the time complexity will be less than O(N)?
b. Present your solution idea as a pseudocode/ python code/ flowchart/ step-by-step instructions/ logical explanation in one-two paragraphs.
c. Write the time complexity of your algorithm.

Q2. You know how binary-search works. Now you are trying to implement it to do other tasks that will take you less than O(n) time to solve. Modify the binary search algorithm so that, if there are duplicates of a number in a sorted array you should be able to tell how many times a number has been duplicated? For example: [1,3,4,5,13,15,16,16,16,16,19,21,21,23], modified_bin_search(16) should return the value 4 since, "16" is present for 4 times in this array. Do that while achieving time complexity less than O(n).

Q3. For different tasks, we often need to search for things. And most of the time, we sort the dataset before performing search operations. Once, you find the following list of numbers.

[2, 5, 1.2, 6.7, 1.7, 9.3, 2.2, 7.7, 0, -4, -5.1, 2, 5, 5.2]

a. To search an element in an unsorted array, we need O(n) time using linear search. Binary search works in O(lgn) time but the array needs to be sorted beforehand which takes at least O(nlgn) time in general. Why would you then sort the array first and then perform binary search instead of just performing linear search?

b. Can you modify count sort so that it may work with negative integers as well?

c. Can you modify count sort so that it may work with the given list?

d. Say, you are working in a system where you need to sort a very big dataset. The memory available to you can barely accommodate the data. You have two options – merge sort & quick sort. Which one should you choose? Why?

e. Construct an array where quick sort fails to work in O(nlgn) time.

## Part 3: Sorting

Q1. 1. Suppose you want to sort an array containing N integers. But every element of the array is the same. Write the asymptotic running times of the following sort algorithms in this case.

a. Selection sort
b. Merge sort

Q2. You are given a list of n integers where the even indices hold numbers in decreasing order and the odd indices hold numbers in increasing order. For example, this is a list of n=7 integers

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Number | 23 | 2 | 19 | 3 | 7 | 11 | 5 |

a. Propose a linear time algorithm to sort the list.
b. Present your algorithm as a code/ pseudocode/ flowchart/ step-by-step instructions.

**Part 4: Recursive Time Complexity + Divide & Conquer Algorithm**

RESOURCES: I have attached some additional resources here:

2. Recursion time complexity HFN.pdf

2.2 Recursion time complexity HFN additional.pdf

Q1. Calculate the time complexity of the following recurrence relations.
[Any method is acceptable as long as steps are shown]

A. $T(n) = 2T(n/2) + 1/n$

B. $T(n) = 625T(n/5) + n^5$

C. $T(n) = T(n/2) + T(n/5) + n$

D. $T(n) = 2T(n/4) + n^2$

Q2. Inspired by the Karatsuba algorithm, a curious CSE student Benjamin decided to modify the algorithm in his own way. For a n digit number, instead of using subproblems of size n/2, Benjamin used subproblems of size n/3. He believes with this modification, he can get a faster algorithm than Karatsuba's. So, for finding product of two n digit numbers A and B, Benjamin splitted A into 3 subproblems ($A_1$ , $A_2$ , $A_3$ ) each with n/3 digits and B into 3 subproblems ($B_1$ , $B_2$ , $B_3$ ) each with n/3 digits. Benjamin wants to write a divide and conquer algorithm for finding the product of A and B from these smaller subproblems. (Assume n is a power of 3)

A. Write A in terms of $A_1$ , $A_2$ , $A_3$ and n. Write B in terms of $B_1$ , $B_2$ , $B_3$ and n

B. Calculate the product AB from your answers in (A)

C. Help Benjamin write the pseudocode of the divide and conquer algorithm

D. Calculate the time complexity of your algorithm and validate whether Benjamin's claim of getting a faster algorithm is true or not.

Q3. Your friend gave you a binary string B (meaning each character is either 0 or 1). He wanted to find out how to calculate the maximum number of consecutive 0s in that particular string. For example,

String: 100100000111 Maximum consecutive 0s: 5

String: 1010101010101 Maximum consecutive 0s: 1

You, as an algorithm enthusiast, know that this can be solved in linear time. However, your friend asked you to propose a Divide and Conquer approach

a) Name a suitable Divide and Conquer algorithm for this task.

b) Explain how you can apply that algorithm in this scenario. Present your idea in a pseudocode/programmable code/Flowchart/step-by-step instructions.

c) Write the time complexity of your algorithm.