CS-430

Homework-1

1) a) Let A be the Array we have which contains the number.

```
for i ← 1   to length[A]-2
    for j ← i+1   to length[A]-1
        for k ← j+1  to  length[A]
            if (A[i]+A[j]==A[k]
            or  A[j]+A[k]==A[i]
            or  A[k]+A[i]== A[j])
                print "found"
            k ← k+1
        j ← j+1
    i ← i+1
```

1b) Algorithm Analysis

```
for i ← 1   to length[A]-2                    c1   (n-1)
    for j ← i+1  to length[A]-1               c2   (n-1)(n-2)
        for k ← j+1  to length[A]             c3   (n-1)(n-2)(n-3)
            if (A[i]+A[j] == A[k]             c4   (n-1)(n-2)(n-3)
            or  A[j]+A[k] == A[i]
            or  A[k]+A[i] == A[j])            c5   (n-1)(n-2)(n-3)
                print "found"                 c6   (n-1)(n-2)(n-3)
            k ← k+1                           c7   (n-1)(n-1)
        j ← j+1                               c8   (n-1)
    i ← i+1
```

Complexity $T(n) = c_1(n+1) + c_2(n-1)(n-2) + c_3(n+1)(n-2)(n-3)$

$+ c_4(n-1)(n-2)(n-3) + c_5(n-1)(n-2)(n-3)$

$+ c_6(n-1)(n-2)(n-3) + c_7(n-1)(n-2) + c_8(n-1)$

$= (c_3+c_4+c_5+c_6+c_7)n^3 + (c_2 + \dots)n^2$

$+ (c_1 + \dots)n - c_1 + \dots$

As we consider that the function is of order $n^2$

so,

**Big O = $O(n^3)$**

1c) Let us assume that we will use merge sort to sort the array as its complexity is $O(n \log n)$.

Merge sort :-

```
Merge (A, P, q, r)
  n1 = q - P + 1
  n2 = r - q
  for i = 1 to n1
      L[i] = A[P + i - 1]

  for j = 1 to n2
      R[j] = A[q + j]

  L[n1+1] = ∞
  R[n2+1] = ∞
  i = 1
  j = 1
  for k = p to r
    if L[i] ≤ R[j]
      A[k] = L[i]
      i = i + 1
```

else A[k] = R[j]

j = j+1

Mergesort (A, p, q, r)

if p < r

$p = \frac{q+r}{2}$

Mergesort (A, p, q)

Mergesort (A, q+1, r)

Merge (A, p, q, r).

Once the Array is sorted, The program to check Sum of pairs is

for i ← 1 to length[A] -3

for j ← i+2 to length[A].

if A[i] + A[i+1] == A[j]

print "Found"

1d) As we know that merge sort follows a recursive approach of time complexity. O(n log n)

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

Algorithm Analysis:

| | Cost | Times |
|---|---|---|
| for i ← 1 to length[A] -3 | $c_1$ | n-1 |
| for j ← i+2 to length[A] | $c_2$ | (n-1)(n-2) |
| if A[i] + A[i+1] == A[j] | $c_3$ | (n-1)(n-2) |
| print "found". | $c_4$ | (n-1)(n-2) |

$$T(n) = c_1(n-1) + c_2(n-1)(n-2) + c_3(n-1)(n-2)$$

$$+ c_4(n-1)(n-2).$$

$$T(n) = (c_2+c_3+c_4)n^2 + (c_1-c_2-c_3-c_4)n + n\log n$$ → for Merge sort

Big O if we consider $n\log n$ & $n$ are less significant sg

$$O(n) = n^2$$

It is more efficient than brute force search.

(2)

Proof :-

Base Case :-

When $k=0$ . k is level.

Binary tree

So, the number of nodes are

$$= 2^0 = 1.$$

When k is k :-

$$= 2^k$$

When k is k+1 :-

$$= 2^{(k+1)}$$

When k is k+1 it is similar to when k is k. So, By Induction we can

say that number of nodes are $2^n$.

3) a)

$2^{n+1} ?= ? O(2^n)$

from definition,

$f(n)$ belongs to $O(g(n))$ if $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$

where $c > 0$ and $n_0 \ge 1$.

So,

$$0 \le 2^{n+1} \le c \cdot 2^n$$

$$0 \le 2^n \cdot 2 \le c \cdot 2^n$$

$$\boxed{c \ge 2}$$

So, with $c = 2$, and $n_0 = 1$.

$$\boxed{2^{(n+1)} = O(2^n).}$$

3b)

$2^{2n} ?= ? O(2^n)$

Let prove or disprove this by limit method.

We know that if

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

$f(n)$ to be Big $O(g(n))$, $\lim_{n \to \infty} \frac{g(n)}{f(n)} = \infty$,

so, $\lim_{n \to \infty} \frac{2^n}{2^{2n}}$

$= \lim_{n \to \infty} \frac{2^n}{2^n \cdot 2^n}$

$= \lim_{n \to \infty} \frac{1}{2^n}$

$= 0$

By definition if $f(n) = O(g(n))$ $\lim_{n \to \infty} \frac{g(n)}{f(n)}$ should be $\infty$

We got 0 instead so

$$2^{2n} \neq O(2^n)$$

4)

a)

```
                                          Cost        Time
int firstDecrease (int *L, int n) {

    for(int i=2; i<=n && L[i] >= L[i-1]; i++)      c1          n

    {

    }

    return i;

}
```

To analyze the function lets write Time complexity interms of n then we will assume the case where we find i and there is some output for $L[i] < L[i-1]$. i will be $n+1$.

As we are talking about the worst case scenario i can take a value of $n+1$.

H(a)

(i) From the definition of bigo $f(n) = O(g(n))$ if $0 \le f(n) \le c \cdot g(n)$ for $n \ge n_0$.

So,

Let prove, $n ?= ? O(i)$

from the cost function above analyzying pseudo code.

$$T(n) = c_1 \cdot n$$

The function is linear with norder of growth

from (i)

$n ?= ? O(i)$

$f(n) = n$

$g(n) = i$

definition

$$0 \le n \le c \cdot (i) \quad -(a)$$

So, $i \propto n$ and can take values: $i \begin{cases} i & \text{when } i \le n \\ n+1 & \text{when } i > n. \end{cases}$

assume the worst case and we get $i = n+1$ when $i > n$.

Substituting $i = n+1$ in (a).

$$0 \le n \le c \cdot (n+1)$$

for $c = 1$, $n > 0$ this value equation always holds true.

$n \leq n+1$

To be precise,

$\boxed{n < n+1}$ for all $n \geq 0$.

From the definition $\boxed{\text{Big}O = O(i)}$

4(b)

ii)

$$E(x) = \mu = \sum x_i P(x_i)$$

so, for random variable

$x_1 = i$, $P(x_1) = \frac{(i-1)}{i!}$

$x_2 = n+1$, $P(x_2) = 1/n!$.

So,

$$= \sum_{i=1}^{2} x_i P(x_i)$$

$$= i \times \frac{(i-1)}{i!} + (n+1) \times \frac{1}{n!}$$

$$\boxed{E(x) = \frac{i(i-1)}{i!} + \frac{(n+1)}{n!}.}$$

4) c) From expectation,

$$E(x) = \frac{i(i-1)}{i!} + \frac{n+1}{n!}.$$

Since we are looking at average big 'O' i can take values from 2 to n. Adding all probabilities would be,

So,

$$E(x) = \sum_{i=2}^{n} \frac{i(i-1)}{i!} + \frac{(n+1)}{n!}$$

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

$$= \frac{1}{0!} + \frac{1}{1!} + \cdots \frac{1}{(n-2)!} + \frac{(n+1)}{n!}$$

$$= 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \cdots \frac{1^{n-2}}{n-2!} + \frac{n+1}{n!}$$

$$= 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \cdots + \frac{1^{n-2}}{n-2!} + \frac{n}{n-1!} + \frac{1}{n!}.$$

$$= 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \cdots \frac{1^{n-2}}{n-2!} + \frac{1^{n-1}}{(n-1)!} + \frac{1}{n!}$$

From tailor series we know that

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} \cdots$$

When $n \to \infty$

Then

$$E(n) = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} \cdots + \frac{1}{n!} + \cdots$$

So,

$$\boxed{E(n) = e.}$$ → Best case

Worst case will be when $i = n+1$;

So,

$$\boxed{E(n) = n}$$

$$\text{Big 'O' Avg} = \left(\frac{n+e}{2}\right)$$

$$= O\left(\frac{n+e}{2}\right)$$

which is

$$= O(n)$$

5) a)

Let us visualize the call of

z ( A, n, 0) ;

Given array A = {1, 2, 3}.

So,
the first call would be

z ( A, 3, 0)

From the function z:

base case when k = n-1 i.e; 2 it will print elements in array.

else it will run

```
for (i= k; i<n ; i++) {
swap(A[i], A[k]) ;

z (A, n, k+1);
swap (A[i] A[k]);
```

So,

n K
z (A, 3, 0) since k=0;
it will run.

```
for (int i=0; i< 3 ; i++ )
{
    swap(A[i], A[0])
    z (A, 3, 1);
    swap(A[i], A[k]);
}
```

Array; A = | 1 | 2 | 3 |  (0 1 2)

for i=0;

swap A[0], A[0]

z (A, 3, 1) ⟶ Recursive call: z (A,3,1); n=3; k=1

k ≠ 2

so,

for int i=1; i<3; i++

swap A[i], A[1]

z (A, 3, 2)

swap A[i], A[1]

when i=1

swap A[1], A[1]

z (A,3,2) → Base case
Print Out (1,2,3) Array
endl

No effect

swap A[1], A[1]

when i=2

swap A[2), A[1]

z (A,3,2)

Print Out (1,3,2) Array
endl

swap A[2], A[1]

A = | 1 | 3 | 2 |  (0 1 2)

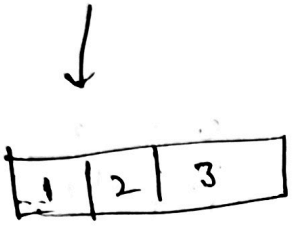
A = | 1 | 2 | 3 |

For i=1;

SWAP A[1], A[0] → A = [2 | 1 | 3]

Z[A, 3, 1] → Recursive call

Swap A[1], A[0]




→ As we see this function prints the array and swapping indexes at 1 & 2 printing the output and reswapping the array, which will print:

1 2 3 endl

2 1 3 endl

(i) print array

(ii) swap 1 & 2 index

(iii) print array

(iv) reswap

print 213endl ←

print 231endl ←

For i=2;

swap A[2], A[0] → A = [3 | 1 | 2]

Z[A, 3, 1) → Recursive call

print 312endl ← (i) print array

(ii) swap 1 & 2 index

(iii) print array

print 321endl ← (iv) reswap

swap A[2], A[0] A = [1 | 2 | 3]

the final output of the function will be

1 2 3 endl

1 3 2 endl

2 1 3 endl

2 3 1 endl

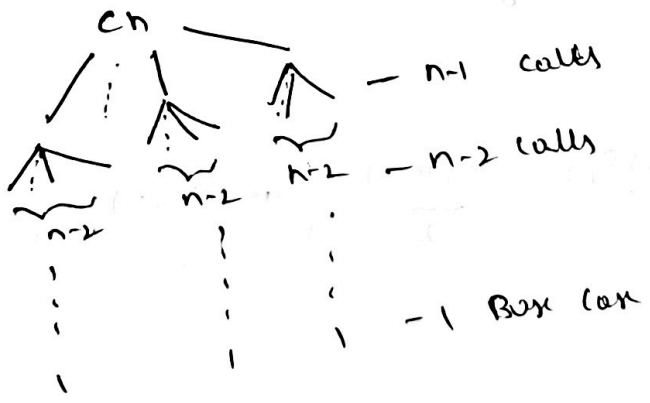
3 1 2 endl

3 2 1 endl

5)b)

We know that the general recursive relation will be in form $T(n) = aT(n/b) + cn$.

if we see each time for a value of i we are making $(n-1)$ calls in the program. So, the calls will be.




So,

$$a = \sum_{k=1}^{n-1} (n-k) = (n-1) + \cdots n-(n-1)$$

$$a = \frac{(n-1)(n)}{2}$$

$$T(n) = \frac{n(n-1)}{2} \cdot T(n-1) + n$$

$T(n-1)$ because each function is reducing its complexity by 1 and solving the problems.

5) c)

$$T(n) = \frac{n(n-1)}{2} T(n-1) + n.$$

Let us write box for $T(n-1)$

$$T(n-1) = \frac{(n-1)(n-2)}{2} T(n-2) + n-1.$$

Substituting;

$$T(n) = \frac{n(n-1)}{2} \left( \frac{(n-1)(n-2)}{2} T(n-2) + n-1 \right) + n$$

$$= \frac{n(n-1)^2(n-2)}{4} T(n-2) + \frac{n(n-1)^2}{2} + n$$

$$T(n-2) = \frac{(n-2)(n-3)}{2} T(n-3) + n-2.$$

Substituting

$$T(n) = \frac{n(n-1)^2(n-2)}{4}\left(\frac{(n-2)(n-3)}{2}\,T(n-3)+n-2\right)$$

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

$$=\frac{n(n-1)^2(n-2)^2(n-3)}{8}\,T(n-3)+\frac{n(n-1)^2(n-2)^2}{2}$$

$$+\frac{n(n-1)^2}{2}+n.$$

For base case similar

If we run k times.

$$=\frac{n(n-1)^2(n-2)^2(n-3)^2\cdots(n-k)}{2^k}\,T(n-3)+$$

$$\frac{n(n-1)^2\cdots(n-k)}{2^k}+\cdots\cdots n.$$

Base Case when $n-k=1$.

$$=\frac{n(n-1)^2(n-2)^2\cdots 1}{2^k}\times 1+\frac{n(n-1)^2\cdots(1)}{2^k}+\cdots n$$

As we are going with Big O ignoring constants.

$$=\frac{n^2(n-1)^2(n-2)^2(1)}{2^k}+n\left(\frac{(n-1)!*(n-2)!*1}{\text{constants}}\right)$$

= Dropping constants as we are going with BigO

$= n(n-1)^2(n-2)^2 \cdots 1 + n(n-1)^2(n-2)^2 \cdots 1$

$+ (n-1)^2 \cdots n(n-1)^2(n-2)^2$

$+ n(n-1)^2 + n$

$= n((n-1)!^2 + (n-1)!^2 + \cdots 1)$

$= n((n-1)! + (n-2)! + \cdots + 1)$

$= n(n-1)! + n(n-2)! + \cdots + n.$

Big-O = $n! + n(n-2)! + \cdots + n.$

We know that $n!$ has a bigO of $n^n$.

$'O(n)' = n^n$

6)

6)a)

<2, 3, 8, 6, 1>

Given to identify pairs when $i<j$ and $A[i] \to A[j]$, $(i,j)$ is an inversion pair.

If we look through the given array

Consider,

2 at index 1 (Assuming index starts at 1)

1 at index 5

So,

(2,1) is an inversion pair

Similarly.

(3, 1)

(8, 1)

(6,1) are inversion pairs.

and

8 at index 3

6 at index 4

So,

(8,6) is an inversion pair.

So,

(2,1), (3,1), (8,1) & (6,1) & (8,6) are inversion pairs.

6)b)

{1, 2, 3, 4, 5... n}

The set will have maximum inversion pairs if the array is in reverse sorted order.

$\langle n, n-1, \ldots 2, 1 \rangle$.

So, if we taken we have

$(n, n-1)$
$(n, n-2)$
$\vdots$
$(n, 1)$

$n-1$ pairs

similarly for $n-1$ we have $n-2$ pairs.

So,
The total number of pairs willbe

$(n-1) + (n-2) + \ldots 1$.

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

The total number of inversion pairs will be $\frac{n(n-1)}{2}$.

6)c) Comparison with insertion sort :-

The number of inversions can effect the insertion sort Complexity if this array is sorted using insertionsort.

If inversion pairs are more the Complexity of insertion sort will be more as everything needs to be rearranged.

So, when the array is in reverse sorted manner it will be the worst case for the insertion sort & if the array's in sorted order it will be best case.

Best Case : $O(n)$
Worst Case : $O(n^2)$

to find inversion pairs as well as insertion sort.

7) a)

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

Let assume the base case, $n=0$.

$$T(n) = \begin{cases} 1 & n=0. \\ T(n-1)+n & n>0. \end{cases}$$

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

We know by,

$T(n-1) = T(n-2) + n-1$

Substituting.

$T(n) = [T(n-2) + n-1] + n.$

By,

$T(n-2) = T(n-3) + n-2.$

Substituting.

$T(n) \{[T(n-3) + n-2] + (n-1)) + n$

$= T(n-3) + (n-2) + (n-1) + n.$

similarly when k times if the Program executes

$$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + \cdots + (n-1) + n$$

Assume the base case,

$n-k=0$

$T(n) = T(n-n) + (n-(n-1)) + (n-(n-2)) + \cdots (n-1)+n$

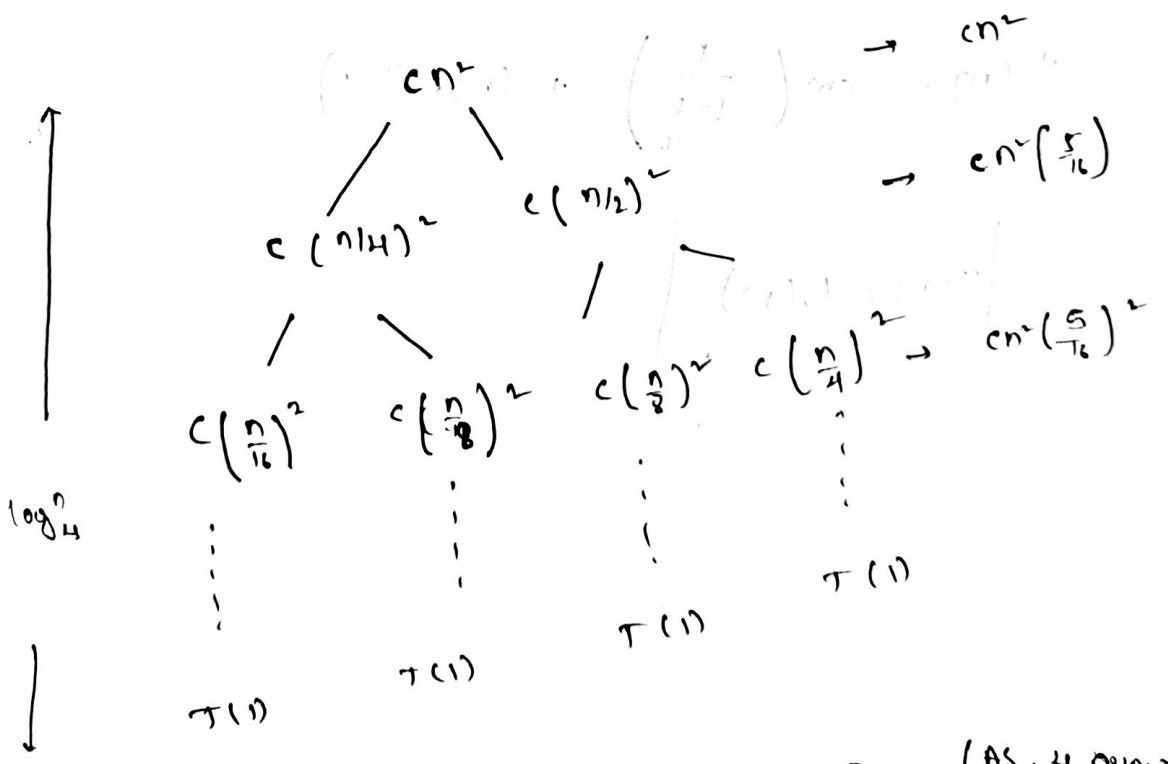
$= T(0) + 1+2+3+4 \cdots (n-1)+n$

$T(n) = T(0) + \frac{n(n+1)}{2}$

$\therefore O(n^2)$

3b) $T(n) = T(n/4) + T(n/2) + n^2$ Let the cost be c for $n^2$

Let's build a recursion tree,




→ Number of levels, $L = \log_4^n$. $L = \log_4^n + 1$ (As, 4 grows rapidly)

Cost of ith level = $\left(\frac{5}{16}\right)^{i-1} cn^2$.

Cost
The ~~cost~~ of all leaves is,

$$2^{\log_4 n} = n^{\log_4 2}$$

$$\Theta(n^{\log_4 2})$$

$$T(n) = cn^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \cdots \left(\frac{5}{16}\right)^{\log_4 n}\right) + \Theta(n^{\log_4 2})$$

$$cn^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \cdots \left(\frac{5}{16}\right)^{n}\right) + \Theta(n^{\log_3 2}).$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$T(n) = cn^2\left(\frac{1}{1-\frac{5}{16}}\right) + \Theta(n^{\log_4 2})$$

$$\boxed{T(n) = O(n^2)}$$