

CS-430

Homework-1

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

for $i \leftarrow 1$ to $\text{length}[A] - 2$

 for $j \leftarrow i+1$ to $\text{length}[A] - 1$

 for $k \leftarrow j+1$ to $\text{length}[A]$

 if $(A[i] + A[j]) == A[k]$

 or $A[j] + A[k] == A[i]$

 or $A[k] + A[i] == A[j]$

 print "found"

X

$k \leftarrow k+1$

$j \leftarrow j+1$

$i \leftarrow i+1$

1b)

Algorithm Analysis

for $i \leftarrow 1$ to $\text{length}[A] - 2$

 for $j \leftarrow i+1$ to $\text{length}[A] - 1$

 for $k \leftarrow j+1$ to $\text{length}[A]$

 if $(A[i] + A[j]) == A[k]$

 or $A[j] + A[k] == A[i]$

 or $A[k] + A[i] == A[j]$

 print "found"

$k \leftarrow k+1$

$j \leftarrow j+1$

$i \leftarrow i+1$

<u>Com-</u>	<u>Time</u>
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	$(n-1)$

$$\begin{aligned}
 \text{Complexity } T(n) = & c_1(n^4) + c_2(n^3)(n^2) + c_3(n^3)(n^2)(n-1) \\
 & + c_4(n^3)(n^2) + c_5(n^3)(n^2)(n-2) \\
 & + c_6(n^3)(n^2)(n-3) + c_7(n^3)(n^2) + c_8(n^3) \\
 = & (c_3 + c_4 + c_5 + c_6 + c_7)n^3 + (c_2 + \dots)n^2 \\
 & + (c_1 + \dots)n - c_1 + \dots
 \end{aligned}$$

As we can see that the function is of Order n^3

so,

$$\boxed{\text{Big } O = O(n^3)}$$

- (i) Let us assume that we will use merge sort to sort the array or its complexity is $O(n \log n)$.

Merge sort =

merge($A[P, Q], R$)

$$n_1 = Q - P + 1$$

$$n_2 = R - Q$$

for $i = 1$ to n_1

$$L[i] = A[P+i]$$

for $j = 1$ to n_2

$$R[j] = A[Q+j]$$

$$L[n_1+1] = \infty$$

$$R[n_2+1] = \infty$$

for $i = 1$ to n_1 do $L[i] = \min(L[i], R[i])$ to choose minimum

$$s = 1$$

for $k = P+1$

if $L[i] \leq R[s]$ then

$$A[k] = L[i]$$

$$i = i + 1$$

$\text{MERGE } A[i:j] = R[t]$

$J = J+1$

Mergesort (A, P, Q, R)

if $P \leq R$

$$P = \frac{Q+R}{2}$$

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 $i \leftarrow 1$ to $\text{length}[A]-3$

for $j \leftarrow i+2$ to $\text{length}[A]$:

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

print "Found"

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

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

Algorithm Analysis:

for $i \leftarrow 1$ to $\text{length}[A]-3$

for $j \leftarrow i+2$ to $\text{length}[A]$

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

print "Found".

Cost

Time

$n-1$

$(n-1)(n-2)$

$(n-1)(n-2)$

$(n-1)(n-2)$

$$T(n) = c_1(n-1) + c_2(n-1)(n-2) + c_3(n-1)(n-2)$$
$$+ c_4(n-1)(n-2).$$

$$T(n) = (c_1 + c_2 + c_3 + c_4)n + (c_1 - c_2 - c_3 - c_4)n + n \log n$$

For merge sort

But if we consider $n \log n$ & n are less significant

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

It is more efficient than brute force search.

(2)

Proof :-

Base Case :-

When $k=0$, this level has 1 node.

Binary tree



So, the number of nodes are

$$= 2^0 = 1.$$

When k is ~~is~~ $=$

$$= 2^k$$

Now consider $k+1$

When this $k+1$:

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

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

Say that number of nodes are $\leq 2^n$.

3) a)

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

from definition,

$f(n)$ belongs to $O(g(n))$ if $0 \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0$,

where c_1 and n_0 are

so,

$$0 \leq 2^{n+1} \leq c_1 \cdot 2^n$$

$$0 \leq 2^n \cdot 2 \leq c_1 \cdot 2^n$$

$$\boxed{c=2}$$

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

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

3b)

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

Let's prove or disprove this by limit method.

We know that if

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

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

so,

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

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

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

$$= 0$$

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

We got 0 instead so

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

A)

a)

int firstDecrease (int *L, int n) {

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

{

 }

~~Cost~~

Time

}

~~Cost~~

Time

To analyze the function let's write time complexity in terms of n then we will assume the cost where we find i and there is some output for $L[i] < L[i-1]$, it will be $n+1$.

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

HQ
 (ii) From the definition of bigo $f(n) = O(g(n)) \Rightarrow \exists c > 0 \exists n_0 \forall n \geq n_0, f(n) \leq c g(n)$

for $n \geq n_0$.

so,

Let prove, $n \in O(i)$

from the cost function above analyzing pseudo code 1

$$T(n) = C_1 \cdot n$$

The function is linear with order of growth

from (i)

$$n \in O(i)$$

$$f(n) = n$$

$$g(n) = i$$

definition

$$0 \leq n \leq c \cdot i \quad (a)$$

so, $i \leq n$ and constate value: $i \begin{cases} i & \text{when } i \leq n \\ n+1 & \text{when } i > n \end{cases}$

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

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

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

for $c=1, n \geq 0$ this value equation always holds

true.

$$n \leq n+1$$

To be precise,

$$n < n+1$$

for all $n \geq 0$.

From the definition

$$\boxed{\text{BigO} = O(i)}$$

H(b)

(ii)

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

so, for random variable

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

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

so,

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

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

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

a) c) From expectation,

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

since we are looking at average big O'

from fake values from 2 ton. Adding all probabilities would be,

so,

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

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

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

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

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

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

From tail series
we know that

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

when $n \rightarrow \infty$

then

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

so,

$$\boxed{E(n) = e.}$$
 → Best case

worst case will be when $i=n+1$

so,

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

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

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

which is

$$= O(n)$$

~~=~~
~~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:

base case when $k = n - 1$ i.e., 2 it will print

elements in array.

else if will run

for ($i = k; i < n; i++$) {

swap($A[i], A[k]$);

$Z(A, n, k+1);$

swap($A[i], A[k]$);

So,

$Z(A, 3, 0)$ 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]);
}
for (int i=0;

```

Array, $A = \boxed{1 \ 2 \ 3}$

swap(A[0], A[0])

$z(A, 3, 1) \rightarrow$ Recursive call: $z(A, 3, 1); n=3, k=1$

$k \neq 2$

so,

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

swap(A[i], A[i])

$z(A, 3, 2)$

swap(A[i], A[i])

when $i=1$

swap(A[1], A[1])

$z(A, 3, 2) \rightarrow$ Base case
print out (1, 2, 3) array
and

swap(A[1], A[1])

when $i=2$

swap(A[2], A[2])

$z(A, 3, 2)$

print out (1, 3, 2) array
and

swap(A[2], A[2])

$A = \boxed{\begin{matrix} 0 & 1 & 2 \\ 1 & 3 & 2 \end{matrix}}$

$A = \boxed{1 \ 2 \ 3}$

For $i=1$,

swap $A[1], A[0]$

$Z[A, 3, 1]$

Swap $A[1], A[0]$



1	2	3
---	---	---

$$A = \boxed{1 \ 2 \ 1 \ 3}$$

total output

Recursive call

$Z[A, 2, 0]$

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

1 2 3 endl

2 3 endl

(i) print array

(ii) swap 1 & 2 index

(iii) print array

(iv) reswap

For $i=2$; print

swap $A[2], A[0] \rightarrow A = \boxed{3 \ 1 \ 2}$

$Z[A, 3, 1]$

→ Recursive call

print

3 1 endl

(i) print array

→ (ii) swap 1 & 2 index

(iii) print array

print

3 2 endl

(iv) reswap

swap $A[2], A[0]$

$$A = \boxed{1 \ 2 \ 3}$$

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

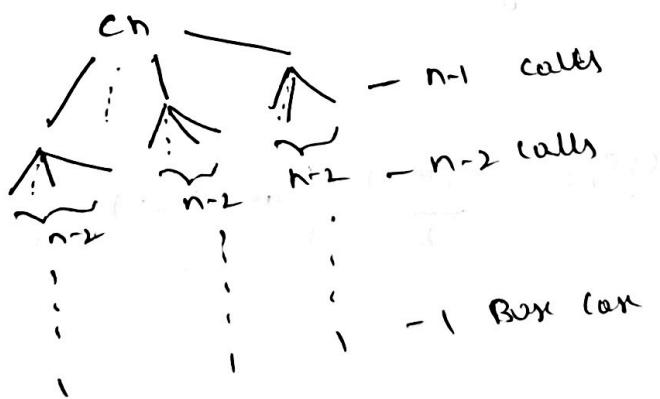
3 2 1 endl

5) b)

We know that the general recursive relation will be in form.

$$T(n) = a + (n/b) + cn.$$

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



so,

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

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

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

X

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

5(c)

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

Now we have to prove that $T(n) = \frac{n(n-1)}{2} T(n-1) + n$

Let us write base case for $T(n-1)$ at $n=1$.

Now we have to prove that $T(n) = \frac{n(n-1)}{2} T(n-1) + n$

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

Substituting;

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

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

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

Substituting

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

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

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

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

for base case similitude

If we run k times.

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

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

Base case when $n-k=1$.

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

As we are going with Big O ignoring constant.

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

= Dropping constants. As we are going with BigO

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

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

$$+ n(n-1) \sim + n$$

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

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

$$= n(n-1)! + n(n-2)! + \dots + n \cdot 1$$

$$\text{Big-O} = n! + n(n-2)! + \dots + n.$$

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

$$\boxed{'O(n)' = n^n}$$



6)

6(a)

$\langle 2, 3, 8, 6, 1 \rangle$

Given to identify pairs when $i < j$ and $A[i] > A[j]$, (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.

X

6(b)

$\{1, 2, 3, 4, 5, \dots, n\}$

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

$2, n, n-1, \dots, 2, 1, 7$

(for N = 7) Total inversion will be

so, if we taken we have:



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

So,

The total number of pairs will be

$$\rightarrow (n-1) + (n-2) + \dots + 1.$$

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

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

b) Comparison with insertion sort :-

The number of inversions ~~is dependent~~ can effect the insertion sort complexity if this array is sorted using insertion sort.

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 is 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 enters

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

Assume the base case,

$$n-k=0$$

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

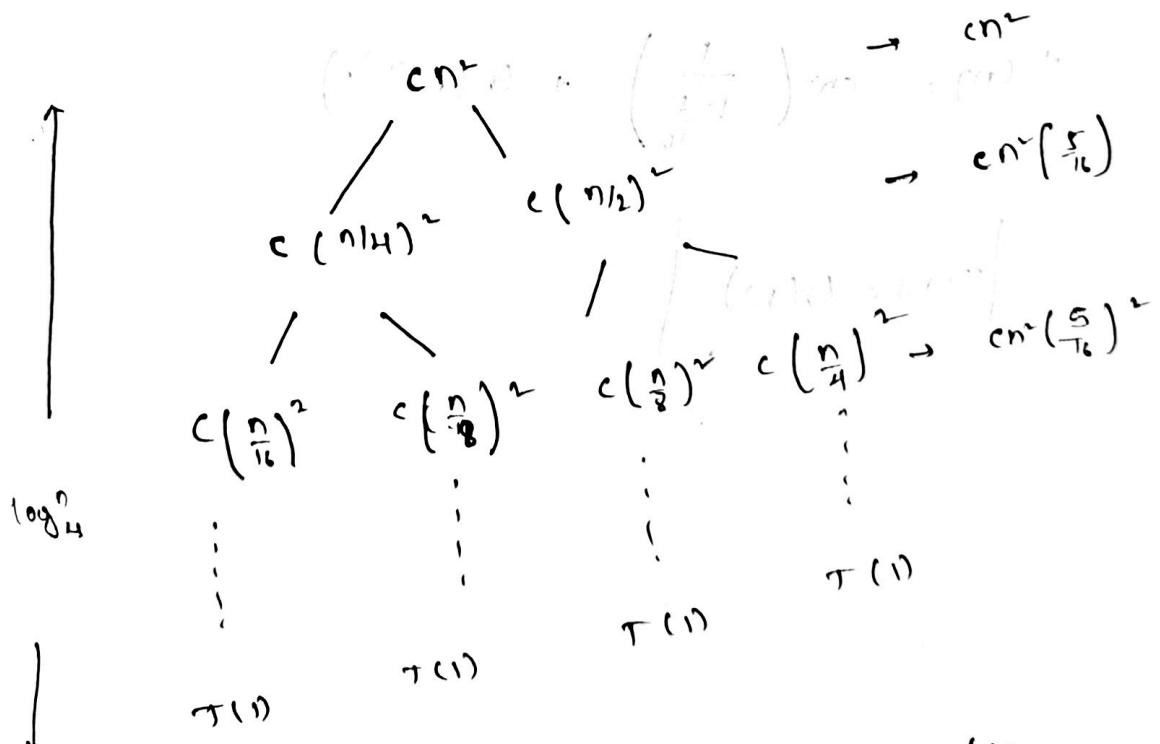
$$= T(0) + 1+2+3+\dots+(n-1)+n$$

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

$\therefore O(n)$

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

Let's build a recursion tree,



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

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

The ^{Cost}
cost of all leaves is,

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

$$\Theta(n \log_4 2).$$

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

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

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

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

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

