Sarthak Durgapal
1961151

PAGE No
DATE

DAA Assignment 1

(24) Recurence relation for recursive binary search.

Sol- Recurrence relation for recursive binary search is :

$T(n) = T(n/2) + 1$, where $T(n)$ is time req. for search of array of size n

Using backward substitution method:

$T(n) = T(n/2) + 1$ $\qquad T(1) = 1$

$T(n) = T(n/2) + 1$ — (i)

$T(n/2) = T(n/4) + 1$ — (ii)

$T(n/4) = T(n/8) + 1$ — (iii)

Putting value from eq. (ii) in (i)

$T(n) = T(n/4) + 1 + 1$ — (A)

Putting eq (iii) in (A)

$T(n) = T(n/8) + 1 + 1 + 1$ — (B)

⋮

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

$\left[\frac{n}{2^k} = 1\right]$

$n = 2^k$

$\log n = \log_2 k$

$\because k = \log_2 n$

$\therefore T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n$

Using $a^{\log_a b} = b$

$T(n) = T(n/n) + \log_2 n$

$T(n) = T(1) + \log_2 n$

$T(n) = 1 + \log_2 n$

Time Comp. $= O(\log_2 n)$

(23) Iterative and recursive pseudocode for binary search

PAGE No
DATE

Iterative:

```
int binarysearch (int arr[], int l, int r, int key)
{
 while (l <= r)
      mid <- l + (r-l)/2
      if arr[mid] == key
           return mid
      if arr[mid] > key
           r <- m-1
      else
           l <- m+1
     End while
     return -1
```

Recursive:

```
int binarysearch (int arr[], int l, int r, int key)
   if r >= l
     mid <- l + (r-l)/2
     if arr[mid] == key
          return mid
     else if arr[mid] > key
        return binarysearch (arr, l, mid-1, key)
        else
          return binarysearch (arr, mid+1, r, key)
     return -1
```

Time complexity	Linear search	Binary search
Best case	O(1)	O(1)
Avg case	O(n)	O(logn)
Worst case	O(n)	O(logn)
Space complexity	O(1)	O(1)

PAGE No
DATE

(22) Divide all sorting algo. into place / Stable / online sorting.

Algo. Name	Place	Stable	Online
① Insertion Sort	✓	✓	✓
② Selection Sort	✓	X	X
③ Bubble sort	✓	✓	X
④ Quick sort	✓	X	X
⑤ Merge sort	X	✓	X
⑥ Heap sort	✓	X	X

(21) Complexity of all sorting algo.

Algo	Space Comp.	Time Complexity: Best Case	Avg Case	Worst Case
① Insertion	$O(1)$	$O(n)$	$O(n^2)$	$O(n^2)$
② Selection	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
③ Bubble	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
④ Quick	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
⑤ Merge	$O(1)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
⑥ Heap	$O(\log n)$, $O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

(2) Time Comp. of - for (i = 1 to n) { i = i * 2 }

Ans = $O(\log_2 n)$

(3) $T(n) = 3T(n-1)$ if $n > 0$, otherwise 1

Sol- $$T(n) = \begin{cases} 3T(n-1), & \text{if } n > 0 \\ 1, & \text{otherwise} \end{cases}$$

Solving by substitution,

$T(n) = 3T(n-1)$ — (i)

$T(n-1) = 3T(n-2)$ — (ii)

Putting (ii) in (i)

PAGE No
DATE

$T(n) = 3[3T(n-2)]$ —— (A)

$T(n-2) = 3T(n-3)$ —— (III)

Putting (III) in A

$T(n) = 3[3(3T(n-3))]$ — (B)

So, $T(n) = 3T(n-1)$

$T(n) = 3[3T(n-2)] = 3^2 T(n-2)$

$T(n) = 3[3(3T(n-3))] = 3^3 T(n-3)$

⋮

$T(n) = 3^n T(n-n)$

$T(n) = 3^n T(0)$

$T(n) = 3^n (1)$

$T(n) = 3^n$

Time Comp = $O(3^n)$

(5)

```
int i=1, s=1;
while (s<=n) {
        i++; s=s+i;
        printf("*");
}
```

$\because 1+2+3 \ldots + k$

$\frac{k(k+1)}{2} > n$

$O(\sqrt{n})$

Time Comp = $O(\sqrt{n})$

(6)

```
void func (int n) {
    int i, count = 0;
    for (i=1; i*i <= n; i++)
        count++;
}
```

$i^2 < n$

$i < \sqrt{n}$

Ans = $O(\sqrt{n})$

(7)

```
for (i=n/2; i<=n; i++)
  for (j=1; j<n; j=j*2)
    for (k=1; k<n; k=k*2)
```

The outermost loop is executing n times and two inner loops will execute $\log n$ times each

$O(n \log^2 n)$

PAGE No
DATE

(8) func (int n) {
if (n == 1) return;
for (i = 1 to n) {
for (j = 1 to n) {
printf ("*"); }}
func (n-3);
}

Sol- $T(n) = T(n-3) + n^2$ — (1)

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

$T(n) = T(n-4) + n^2 + (n-1)^2$

$T(n) = T(n-5) + n^2 + (n-1)^2 + (n-2)^2$

$\vdots$

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

$T(n-k) = 1 \qquad k = n-1$

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

(9) void func (int n) {
for (i = 1 to n) {
for (j = 1; j <= n; j = j+1)
printf ("*"); }
}

Sol- i = 1 n times

i = 2 1, 3, 5, ... $\frac{n}{2}$ i = 3 1, 4, 7 ... $n/3$

$T(n) = \left(n + \frac{n}{2} + \frac{n}{3}\right)$

$T(n) = O(n \log n)$

PAGE No
DATE

(10) The Func, n^k and a^n $k>=1$ and $a>1$

Sol- The asymptotic relationship b/w the functions n^k and a^n is

$n^k = O(a^n)$

$f(n) \leq c \cdot g(n)$ for Big-Oh

So, $n^k \leq c \cdot (a^n)$

if $k=1$, $a=2$, $c=1$

$n^k \leq c \cdot a^n$

$n \leq 2^n$

For all values of $n \leq 2^n$ and $c>=1$, relation will hold true.

(11)

```
Void fun (int n) {
    int j=1, i=0
    While (i<n)
    {
        i = i+j;
        j++;
    }
}
```

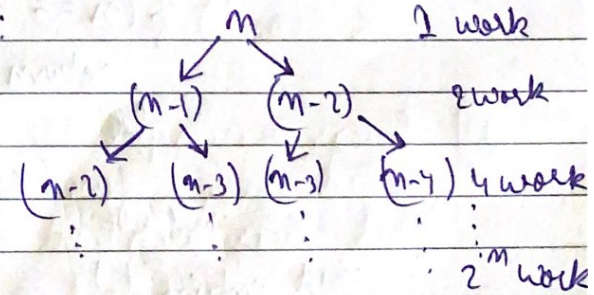
0, 3, 6, 10, 15 ..... n

k^{th} term $= k\left(\frac{k+1}{2}\right) = \frac{k^2+k}{2}$

$T = O(\sqrt{n})$

(12) Recurrence relation for the recursive functions that prints Fibonacci series. $T(n) = T(n-1) + T(n-2) + 1$

Sol- Solving it using tree method:



So, Time comp, $T(n) = 1+2+4+\ldots+2^n$

GP, where $a=1$ $r=2$

$Sum = a\left(\frac{r^{n+1}-1}{r-1}\right) = 1\left(\frac{2^{n+1}-1}{n-1}\right)$

$2^{n+1} = 1$

Time comp $= O(2^{n+1}) = O(2^n \cdot 2)$

Ans: $O(2^n)$

(13) Write programs which have comp:

(i) $n(\log n)$

```
void function(){
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j=j*2) {
                printf("Hello");
        }}}
```

(ii) n^3

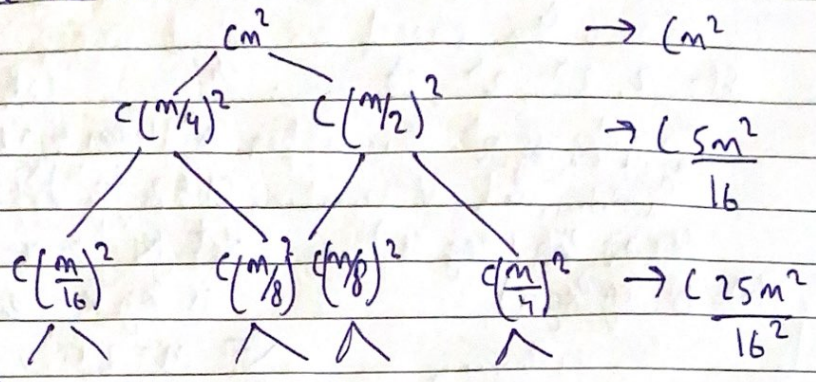
```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        for (int k=0; k<n; k++) {
            print("*");
    }}}
```

(iii) $\log(\log n)$

```
void fun(int n)
{
    if (n == 2)
        return 1;
    else
        fun(sqrt(n));
}
void main()
{
    fun(100);
}
```

(14) Solve $T(n) = T(n/4) + T(n/2) + cn^2$

Sol Solving by tree method



So, $T(n) = cn^2 + c\frac{5n^2}{16} + c\frac{25n^2}{256} + \dots$

$$T(n) = c\left[n^2 + \frac{5}{16}n^2 + \frac{25}{256}n^2 + \dots\right]$$

$$T(n) = cn^2\left[1 + 5/16 + 25/256 + \dots\right]$$

This is a gp with $a = 1$, $r = \frac{5}{16}$

if $r < 1$, Sum of GP $= \frac{a}{1-r}$ $\Rightarrow$ $T(n) = cn^2\left[\frac{1}{1-\frac{5}{16}}\right]$

$$T(n) = cn^2\left[\frac{16}{11}\right] \Rightarrow \boxed{T(n) = O(n^2)}$$

(15)
```
int fun (int n) {
    for (int i=1; i<=n; i++) {
        for (int j=1; j<n; j+=i) {
            some O(1) task
        }
    }
}
```

Sol - $n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \dots$ (K terms)

$K = \log_2 n$

$n\left(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4} \cdots \frac{1}{n}\right) \Rightarrow n(\log n)$

$\boxed{T(n) = O(n \log n)}$

PAGE No
DATE

(16) for (int i=2; i<n; i = pow(i,k))
{ //o(1)
}

Sol- $2, 2^k, 2^{k^2}, 2^{k^3} \ldots n$ GP, $a = 2$ $r = 2^k$

k^{th} term $= ar^{k-1}$

$n = 2(2^k)^{k-1}$

let $k^{k-1} = x$

$k \log_2 k = \log x$

$\Leftarrow n = 2x$

$\log_{2^k} n = x \log_2 2$

$x = \log_2 n$

$\log x = \log(\log n)$

$T(n) = O(\log \log n)$

(18) Arrange in increasing order of growth.

(a) $O(100) < O(\log \log n) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n)$
$< O(n^2) < O(2^n) < O(2^{2n}) < O(4^n) < O(\log n!) < O(n!)$

(b) $2(2^n), 4n, 2n, 1, \log(n), \log(\log n), \sqrt{\log(n)}, \log 2n, 2 \log n,$
$n, \log(n!), n!, n^2, n(\log n)$

Sol $O(1) < O(\log \log n) < O(\log n) < O(\log 2n) < O(2 \log n)$
$< O(n) < O(n \log n) < O(\log n!) < O(2n) < O(4n) < O(n^2) <$
$O(n!) < O(2(2^n))$.

(c) $O(96) < O(\log n) < O(\log_2 n) < O(5n) < O(n \log_{10} n)$
$< O(n \log_2 n) < O(8n^2) < O(7n^3) < O(\log n!) < O(n!)$
$< O(8^{2n})$

(19) Linear search pseudocode to search an element in a sorted array with minimum comparisions.

Sol-
```
void linearsearch(int arr[], int n, int key)
    for i <- 1 to n
        if arr[i] == key
            return i
    return -1
```

(20) Pseudo code for iterative insertion sort :

```
void insertionsort ( int *arr, int n)
    for i<-1 to n
        temp = arr[i]
        j <- i-1
        while j>=0 AND arr[j] > temp
            arr[j+1] = arr[j]
            j <- j-1 ;
        End while
        arr[j+1] <- temp
```

∵ (temp = arr[j+1])

Pseudo code for recursive insertion sort :

```
void insertionsort ( int *arr, int n)
    if n<=1
        return
    insertionsort (arr, n-1)
    last <- arr[n-1]
    j <- n-2
    while j>=0 AND arr[j] > last
        arr[j+1] = arr[j]
        j--
    End while
```

Insertion sort is called online sorting because it processes its input piece by piece i.e., in the order that the input is fed to the algorithm without having the entire input available from the start.

Other sorting algo. are diff from insertion sort because unlike other algo. insertion sort does not waits and as soon as it encounters the elements, it sorts them.

PAGE No
DATE

① Asymptotic notations are methods or languages using which we can define the running time of algorithm based on input size.

To represent this upper and lower bounds we need some kind of syntax, this is represented in the form of function $f(n)$.

Logarithmic — $\log n$
Linear — n
Quadratic — n^2
Polynomial — n^2
Exponential — a^n