the parent pointer may not lead back to the source node if a zero length cycle exists.

relax (a,t)

(a,a) $d[a] \geq d[a] + w(a,t)$

$1 \geq 1+0$

$1 \geq 1$ $d[a] = 1$ $\pi[a] = a$

MT 7(A)

given array A[n] of natural no. TC to find max sum sub array

Sol: O(1) bcz all elements are natural number Hence array itself will be subarray with maximum sum

---

Bit Algorithm

Important points

→ Java does not support unsigned no.

→ Signed no's are represented using 2's comp.

→ Ex: -2 rep: Suppose 4 bits are used to rep no.

$2^4 - 2 = 14 = 1110$

-5 $2^4 - 5 = 11 = 1011$

Bit wise Operator

&, |, ~, ^, >>, <<

↓ bit wise Complement ↓ bit wise XOR

>> → right shift $\equiv \left\lfloor \frac{x}{2^k} \right\rfloor \equiv x >> k$

<< - left shift $\equiv x * 2^k \equiv x << k$

Q Check whether kth bit from right is set or not.

Sol: i/p n = 5 101

k = 1 o/p - True

k = 2 o/p = False

Concept: generate a no in which only kth bit from right is set and perform bit wise & b/w generated no. and given no.

```
if (num & (1 << (k-1)) != 0)
    print True
else print False
```

Q Check for power of 2.

Sol: i/p = 4 o/p - True

i/p = 10 o/p → False

i/p = 0 o/p → False

Method - 1 Keep dividing no by 2, if at any step (num%2 != 0) given num is not power of 2

Method - 2 !(num & (num-1))

↳ return 0 if power of 2
return ≠0 if not power of 2

```
if (!(x & (x-1)))          if (x && !(x & (x-1)))
    print True                 print True
else                       else
    print False                print False
```

Q Counting no of set bits in a given integer.

Sol: Standard algorithm

Brian Brian Kernighan algorithm Unset right most set bit

TC → O(no of set bits)

```
x = 26   11010
int res = 0
While (x > 0)
{
    x = x & (x-1)
    res++
}
return res;
```

```
11010
11000
10000
00000
```

Q given an array arr[1...n] of n integers in which all elements appear even no of times except for one element which appear odd times. find that no element. (elements which occurs even no of times will cancle out each other)

arr[] = {3, 3, 3, 4, 5, 4, 5}

property of XOR $x \oplus 0 = x$

$x \oplus 1 = \bar{x}$

$x \oplus y = x\bar{y} + \bar{x}y$

$(x \oplus y) \oplus z = x \oplus (y \oplus z)$

```
res = 0
for (int i = 0; i < n; i++)
    res = res ^ arr[i]
```

Q Generating power set of a set?

Sol. Str = "ABC" ← given set

Pow(str) = {"", "A", "B", "C", "AB", "AC", "BC", "ABC"}

We can find the size of set. Say n

If we generate binary no's from 0 to $2^n - 1$

Every binary no. corresponds to a set subset of given set n = 3 0 to 7

```
000 → ϕ
001 → A
010 → B
011 → AB
100 → C
101 → AC
110 → BC
111 → ABC
```

given size of set n

Calculate $(2^n - 1)$

```
int count = (1 << n)        // generate no from 0 - (2^n - 1)
for (int i = 0; i < count; i++)      // loop through every bit in no. only
    for (int j = 0; j < n; j++)
        if ((i & (1<<j)) != 0)       // check bit is set or not
            print(str[j])            // the jth element from right in set is present in the subset or not
```

Exercise

Q given a number in binary representation find longest span of 1's.

Done

Q given an array of n integers, in which all elements appear even no of times except for 2 elements which appear odd no of times. find these 2 elements.

Mathematics

finding no of digits in a number

```
While (n != 0)
{
    count++;
    n = n/10;
}
```

```
countDigit(int n)
if (n == 0)
    return
else
    1 + countDigit(n/10)
```

use logarithmic

count = $\lfloor \log_{10}(n) + 1 \rfloor$

Recursion

We solve a problem assuming s/n's to smaller subproblems are available.

Q given a number n, print no's from n to 1 without using loop.

```
printNum(int n)
{
    if (n == 0)
        return
    else
        print(n)
        printNum(n-1)
}
```

tailed recursion

```
print 1 to n
{
    printNum(int n)
    if (n == 0)
        return
    else
        printNum(n-1)
        print(n)
}
```

Advantage of tailed recursion

Compiler optimization Recursive → (Convert) Non-recursive (space saved)

Q given a rope of length n and three values a, b, c we need to make maximum pieces such that every piece has length in set {a, b, c}, & no piece is wasted.

i/p n = 5 a = 1, b = 2, c = 3

o/p 5

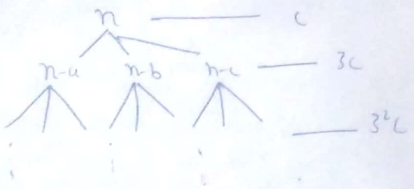
i/p n = 5 a = 2, b = 4, c = 6

o/p : -1

```
int getMax(int n, int a, int b, int c)
{
    if (n < 0)
        return -1
    if (n == 0)
        return 0

    int A = getMax(n-a, a, b, c)
    int B = getMax(n-b, a, b, c)
    int C = getMax(n-c, a, b, c)
int res = imax(A, B, C)
    if (res == -1)
        return -1
    else
        return res+1
}
```

TC $T(n) = T(n-a) + T(n-b) + T(n-c) + C$

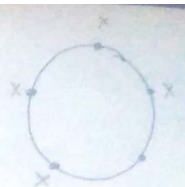


$TC = c + 3c + 3^2c + \cdots + 3^n c$

$= c(1 + 3 + 3^2 + \cdots + 3^n)$

$= O(3^n)$

Q. Josephus Problem



c/p n = 5 k = 2

o/p → 3

n people standing in a circle, in every iteration kth person is killed. We need to find luck place

$$JP(n,k) = (JP(n-1,k) + k - 1) \% n + 1$$

$JP(1,k) = 1$

position of the person who will hold the given n posts

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

$O(n)$

Q. Printing all the subsets of a set

$T(n) = 2T(n-1) + C$ (recursion with n-1 people)

$O(2^n)$

Set = "ABC"

Subsets = {"", "A", "B", "C", "AB", "AC", "BC", "ABC"}

```
void printSubset(string str, int index = 0, string currSubset = "")
{
    int n = str.length()
    if (index == n)
        print(currSubset)
        return
    printSubset(str, index+1, currSubset)
    printSubset(str, index+1, currSubset + str[index])
}
```

Q Exercise: given an integer set and a number print all subsets whose sum is equal to given number

i/p: arr[] = {2, 3, 9, 5, 4, 9}, n = 7

o/p: {2, 5} {3, 4}

Lec 8

Q Tightest lower bound on the number of comparisons ; in the worst case, of comparison based sorting is of the order of

$O(n \log n)$

Sol:- Given a list of n distinct numbers $n!$ permutations are possible, out of which only one is sorted

Sorting algorithm must gain enough information from comparisons to identify the correct permutation.

If algorithm always completes after $f(n)$ steps, then it can not analyze (distinguish) more than $2^{f(n)}$ cases (bcz when comparing, each comparison has only 2 possible outcomes) → either swap / no swap

Hence $2^{f(n)} \geq n!$

$\log 2^{f(n)} \geq \log n!$

$f(n) \geq \log n! \rightarrow f(n) = \Omega(n \log n)$

Q You have to sort 1 GB of data with only 100 MB of available main memory. Which sorting technique will be most appropriate?

Sol:- The data can be sorted using any external sorting technique which uses merging technique

1. Divide 1GB data into chunks of 100MB
2. Sort each chunk group and write them to disk
3. Load 10 items from each group into main memory. Output smallest item from main memory to disk
4. Load next item from the group whose item was chosen
5. loop 4 step until all items are not outputted

Sort nearly sorted array

Given an array of n elements, where each element is at most k away from its correct position in sorted array.. Device an algorithm that sort in $O(n \log k)$ time.

Insertion Sort

i/p: arr[] = {6, 5, 3, 2, 8, 10, 9}
k = 3

o/p: arr[] = {2, 3, 6, 5, 8, 9, 10}

```
for (i=1; i<n; i++)
    arr[i]
    key = arr[i] ;  j = i-1
    while (j>=0 && arr[j] > key)
        arr[j+1] = arr[j]
        j = j-1

    arr[j+1] = key
```

While loop will run at most k times for each element. $T(\rightarrow O(nk)$

Efficient method → Heap data structure

1) Create min Heap of size k+1 with first k+1 elements. → O(k) time

2) One by one remove min element from heap, put it in result array. And add a new element to heap from remaining elements. delete & insert

insertion

$T(\rightarrow O(k) + \cancel{n \log k} + (n-k) \log k$

$T(\rightarrow O(n \log k)$

Segregating numbers

Q given arr[1..n] consists of only +ve & -ve no's. Segregate no's having same sign altogether.

Sol:- We can use partition algo of quick sort.

$T(\rightarrow O(n)$

# of comparisons = n-1