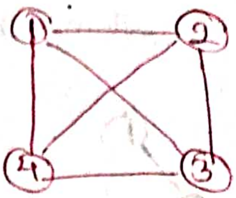
Spanning tree

1. Spanning tree
2. prims MST
3. kruskals MST

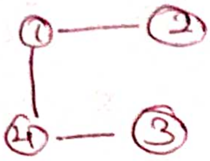


$G = (V, E)$
$n = |V|$
$e = |E|$

${}^6C_3$

$S \subseteq G$
$S = (V', E')$
$|V'| = |V|$
$|E'| = |V| - 1$

Eg:-

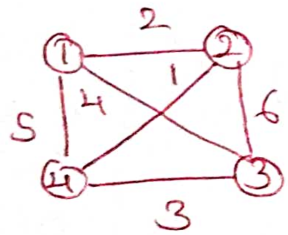


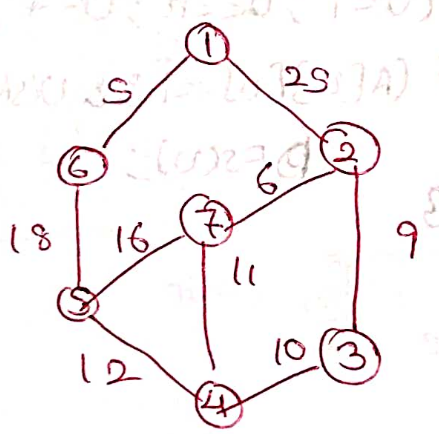
→ figure should be closed

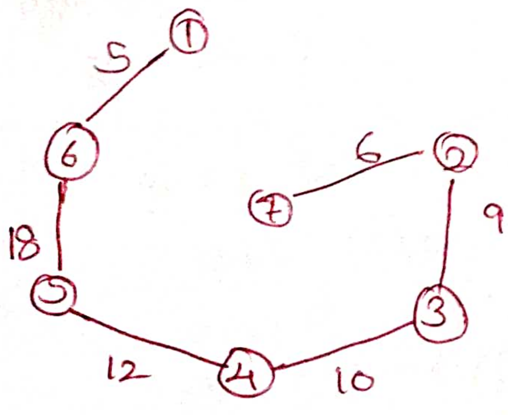
↳ ${}^{|E|}C_{|V|-1} - 4$

↳ ${}^{|E|}C_{|V|-1}$ − cycles $= {}^6C_3 - 4 = 16$

Minimum cost spanning tree







Total cost

$5 + 18 + 12 + 10 + 9 + 6 = 60$

$(|V| - 1)(|E|)$

$ne = n \times n$

$= O(n^2)$

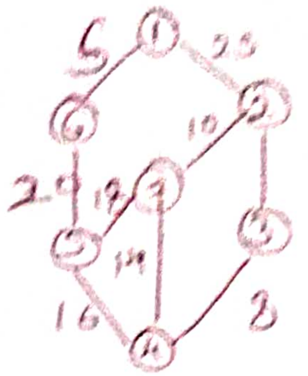
If we use heap

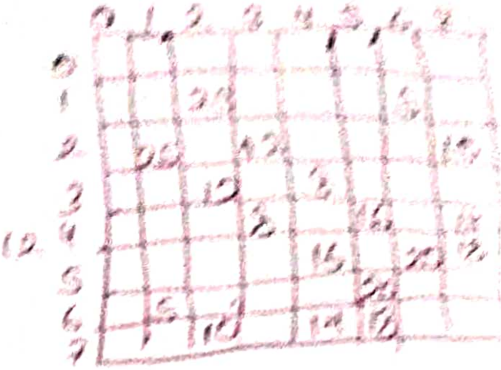
$(|V| - 1) \log |E|$

$O(n \log n)$

Prim's Program

first find minimum edge and update





near

~~–~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~

near

0	1	2	3	4	5	6	7
–	-	-	-	-	-	-	-

repeating steps

[2][1] = 25

[5][6] = 20 ~ min

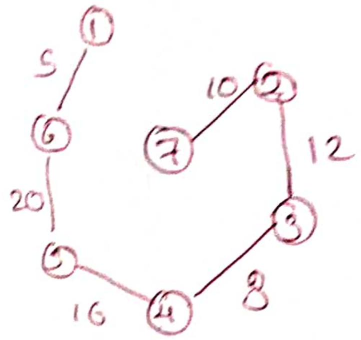
near [5] = 0

t =

	0	1	2	3	4	5
0	1					
1	6					

near

0	1	2	3	4	5	6	7
	0	~~1~~	6	6	6	[illegible]	0 6



near

0	1	2	3	4	5	6	7
	0	1	6	5	0	0	5

k=5

t

	0					
0	1	5	4	3	2	2
1	6	6	5	4	3	7

near

–	0	1	4	0	0	0	4

k=4

k=3 near

0	1	2	3	4	5	6	7
–	0	3	0	0	0	0	4

k=2

Kruskal's

Arrange nodes as per less distance and when cycle comes don't include that edge

time $(|V|-1)(|E|)$ | By heap

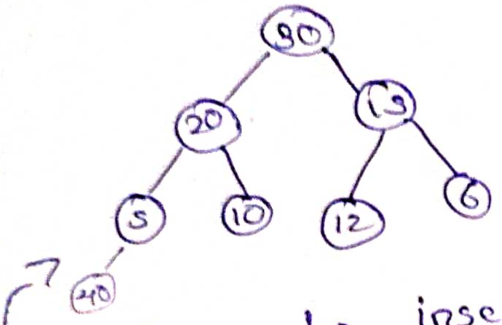
$ne = n \times n$ | $O(n \log n)$

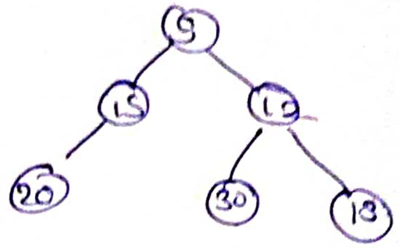
$O(n^2)$

Binary heap

→ heap is a complete binary tree. Mostly represented using arrays. height is log n

max heap   min heap





Procedure to insert   ele = 40

$\frac{8}{2} = 4$

30	20	15	5	10	12	6	40		
1	2	3	4	5	6	7	8	9	10

$\frac{4}{2} = 2$

30	20	15	40	10	12	6	5		

$\frac{2}{2} = 1$

30	40	15	20	10	12	6	5	

40	30	15	20	10	12	6	5	

ele 35

```
void insert(int A[], int n)
{ int temp, i=n;
  temp=A[n];
  while(i>1 && temp>A[i/2])
  { A[i]=A[i/2];
    i=i/2; }
  A[i]=temp; }
```

Delete in Max heap :- Always delete root element

```
Void delete (int A[] ,int n) {
    int x, i, j;
    x = A[n];
    A[i]= A [n] ;
    i=1, j= 2*i
    while (j<n-1 ) {
      if (A[j+1] > A[i]) j=j+1;
      if (A[i] < A[j]) { swap[A[i], A[j]);
                        i= j
                        j =2*j ; }

    else   break; }
      A[n]=x;
    }
```

Heap sort

1. Create heap of 'n' elements . nlog n
2. Delete 'n' ele Oneby 1 nlog n

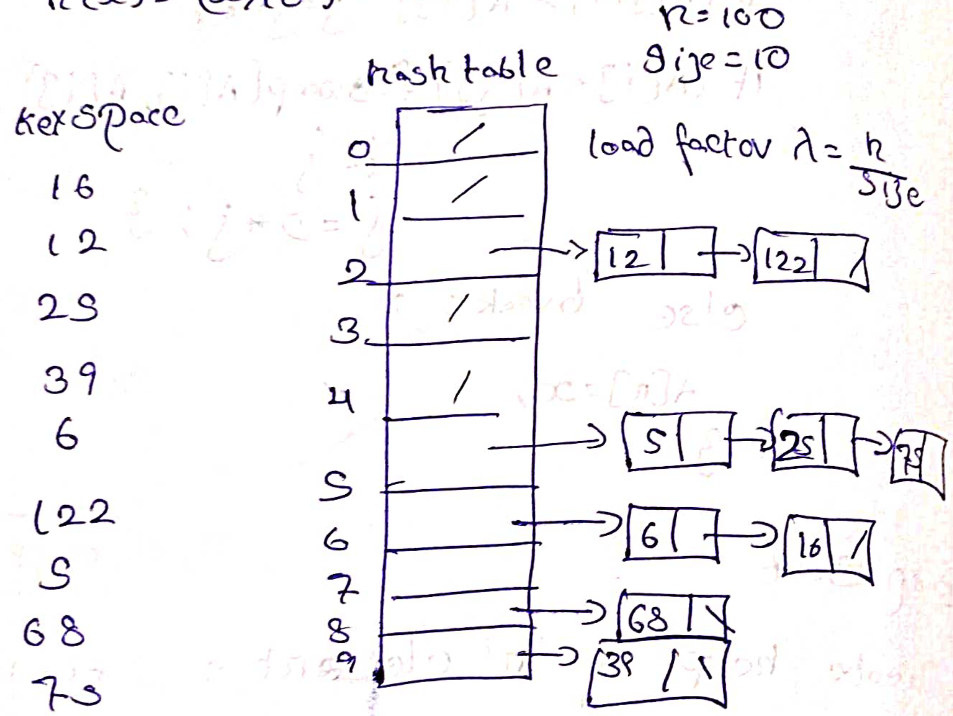
   heapsort 2nlogn = O (nlogn

Hashing techniques O(1)

-> linear Search O(n)
-> Binary Search O(logn) -> bot for Sorting extra work
-> Hasing Techique -> O(1) -> Storage problem

Chaining
keys : 16, 12, 25, 39, 6, 122, 5, 68, 75

$h(x) = (x \% 10) \% 10$



Avg Successful Search $t = 1 + \frac{\lambda}{2}$

Avg unsuccessful Search $t = 1 + \lambda$

Double hashing

$h1(x) = x \% 10$

$h2(x) = 7 - (x \% 7)$

$h'(x) = (h1(x) + i * h2(x) \% 10)$ where $i = 0, 1, 2, \ldots$

linear Probing

$h(x) = x \% 10$

$h'(x) = (h(x) + f(i)) \% 10$

where $f(i) = i$

$i = 0, 1, 2$

26
30
45
23
25
43
74

0	30
1	29
2	
3	23
4	43
5	45
6	26
7	25
8	74
9	19

Delete is very hard

loading factor $\lambda = \frac{n}{size}$

In generel $\lambda \leq 0.5$

Avg. Successful Search $t = \frac{1}{\lambda} \ln\left(\frac{1}{1-\lambda}\right)$

Avg unsuccessful Search $t = \frac{1}{1-\lambda}$

Quadratic Probing

$h'(x) = (h(x) + f(i)) \% 10$ where $f(i) = i^2$

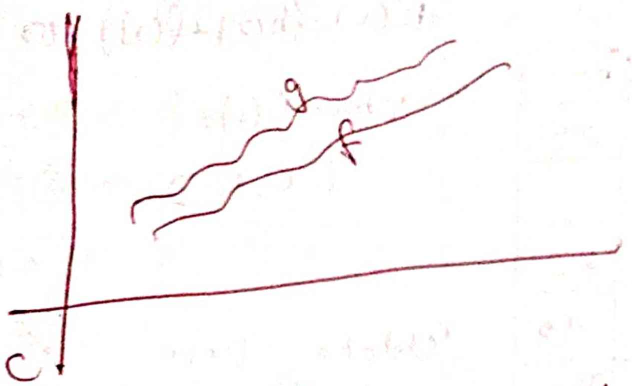
$i = 0, 1, 2$

Avg. Successful Search

$-\frac{\log_e(1-\lambda)}{\lambda}$

Avg unsuccessful Search

$\frac{1}{1-\lambda}$



$f(x) < c \cdot g(x)$

$x > n_0$

Time Complexity

i	p
1	1
2	1+2
3	1+2+3
4	1+2+3+4

1. p=0

```
for(int i=1; p<=n; i++){
        p=p+i;}
```

Ass

$\frac{k(k+1)}{2} = n$ $\qquad k = 1+2+3 \ldots + k$

$k = n$ $\qquad O(\sqrt{n})$

$k = \sqrt{n}$

2. Analysis of if & while loop

```
while (m!=n)
  if (m>n)
     m=m-n;
  else
     n=n-m
```

m=16	n=2
14	2
12	2
10	2
8	2
6	2
4	2
2	2

$\frac{n}{2}$ $O(\frac{n}{2})$

min $O(1)$

max $O(n)$

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \ldots 2^n < \ldots < n^n$

Asymptotic notations

O big-oh upper bound $f(n) = O(g(n))$ if $\exists$ +ve const c & n_0 such that $f(n) \le c \times g(n)$ $\forall\, n \ge n_0$

Ω big-omega lower bound

Θ theta Average bond

Properties of Asymptotic notations

- if $f(n)$ is $\Omega(g(n))$ then $a * f(n)$ is $\Omega(g(n))$

- Reflexive $f(x)$ $f(x)$ is $O(f(n))$

- symmatic $\Theta(g(n))$ for $f(n)$
  $\Theta(f(n))$ for $g(n)$

- Transpose symmetric
  if $f(n) = O(g(n))$ then $g(n)$ is $\Omega(f(n))$

Searching linear search $B(n) = O(1)$
$W(n) = O(n)$

Average case $= \dfrac{\text{all possible case time}}{\text{no. of cases}}$ $A(n) = \dfrac{n+1}{2}$

Avg time $= \dfrac{1+2+3+\ldots+n}{n} = \dfrac{n+1}{2}$

Recurrence relations

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

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

$T(n)$ — n

n $\quad T(n-1)$ — $n-1$

$n-1$ $\quad T(n-2)$ → $n-2$

$\vdots$

$T(1)$

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

```
void Test(int n)
  if (n>0) {
  for(int i=1; i<n; i=i*2)
    Printf("%d", i)
  }
  Test(n-1);
  } }
```

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

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

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

$\log 1 + \log 2 + \dots \log n$

$1 + \log(n(n-1)(n-2) \dots 1)$

$1 + \log n! \rightsquigarrow O(n \log n)$

$T(n) = T(n-1) + 1$ — $O(n)$

$T(n) = T(n-1) + n$ — $O(n^2)$

$T(n) = T(n-1) + \log n = O(n \log n)$

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

$$T(n) = \begin{cases} 1 & n=0 \\ 2T(n-1)+1 & n>0 \end{cases}$$

$T(n)$ — Algo Test(int n){

if (n>0){

1 — Printf("%d", n);

$T(n-1)$ → Test(n-1);

$T(n-2)$ Test(n-1); }

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

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

$= 2^2 T(n-2) + 2 + 2^0$

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

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

$n - k = 0$

$n = k$

$2^n + 2^{k+1} - 1$

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

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

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

$T(n) = 2T(n-1) \cancel{+} n - O(n2^n)$

Master theorem for decreasing function

$T(n) = aT(n-b) + f(n)$

$a > 0$ $b > 0$ & $f(n) = O(n^k)$ where $k \geq 0$

Case

i. if $a < 1$ $O(n^k)$ $O(f(f(n))$

ii if $a = 1$ $O(n^{k+1})$ $O(n * f(n))$

iii if $a > 1$ $O(n^k a^{n/b})$

Dividing fun :-

$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2)+1 & n>1 \end{cases}$$

T(n) Test (int n){
if (n>1){
Printf ("%d", n);
$T(n/2) \longrightarrow$ Test(n/2)

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

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

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

$O(\log n)$

$$\frac{n}{2^k} = 1$$

$$2^k = n$$

$$k = \log n$$

$$T(n) = \begin{cases} 1 & n=1 \\ 2(T(n/2)+n & n>1 \end{cases}$$

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

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

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

$$T(1 = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$\log = \log n$

$O(n\log n)$

Master theorem for dividing functions

$T(n) = aT(n/b) + f(n)$ $a \geq 1$; $b > 1$

$f(n) = \theta(n^k \log^p n)$

① . $\log_b a$ ② k

Case 1 : if $\log_b a > k$ then $\theta(n^{\log_b a})$

Case 2 if $\log_b a = k$

if $p > -1$ $\theta(n^k \log^{p+1} n)$

$p = -1$ $\theta(n^k \log\log n)$

$p < -1$ $\theta(n^k)$

Case 3 if $\log_b a < k$

if $p \geq 0$ $\theta(n^k \log^p n)$

if $p < 0$ $O(n^k)$

Root functions

$$T(n) = \begin{cases} 1 & n = 2 \\ T(\sqrt{n}) + 1 & n > 2 \end{cases}$$ $n = 2^m$

$T(n) = T(n^{1/2}) + 2$ $m = \log_2 n$

$T(n) = T(n^{1/2^k}) + k$

let $n = 2^m$

$T(2^m) = T(2^{\frac{m}{2^k}}) + k$ $k = \log \log_2 n$

$\frac{m}{2^k} = 1$ $\theta(\log \log_2 n)$

$k = \log_2 m$