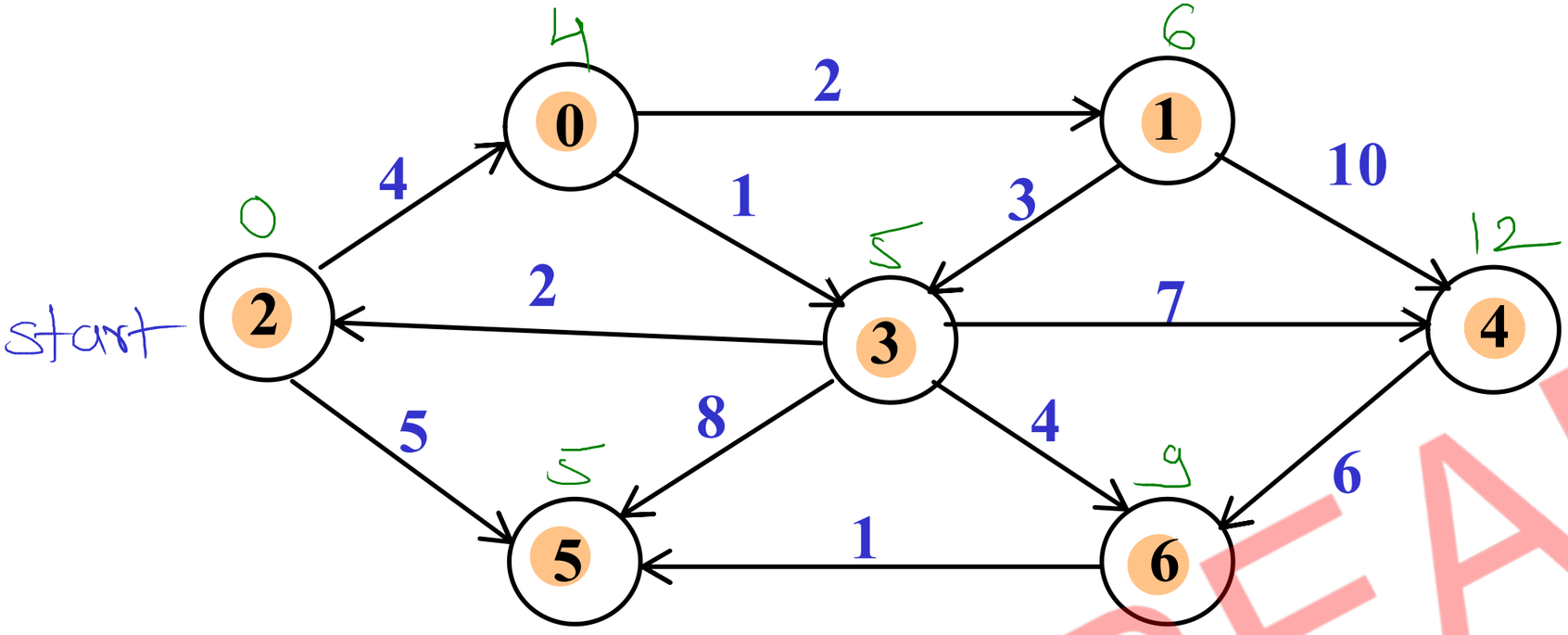
Djkshtra's SPT



	D
0	4
1	6
2	0
3	5
4	12
5	5
6	9

	D
0	4
1	∞
2	0
3	∞
4	∞
5	5
6	∞

	D
0	4
1	6
2	0
3	5
4	∞
5	5
6	∞

	D
0	4
1	6
2	0
3	5
4	12
5	5
6	9

	D
0	4
1	6
2	0
3	5
4	12
5	5
6	9

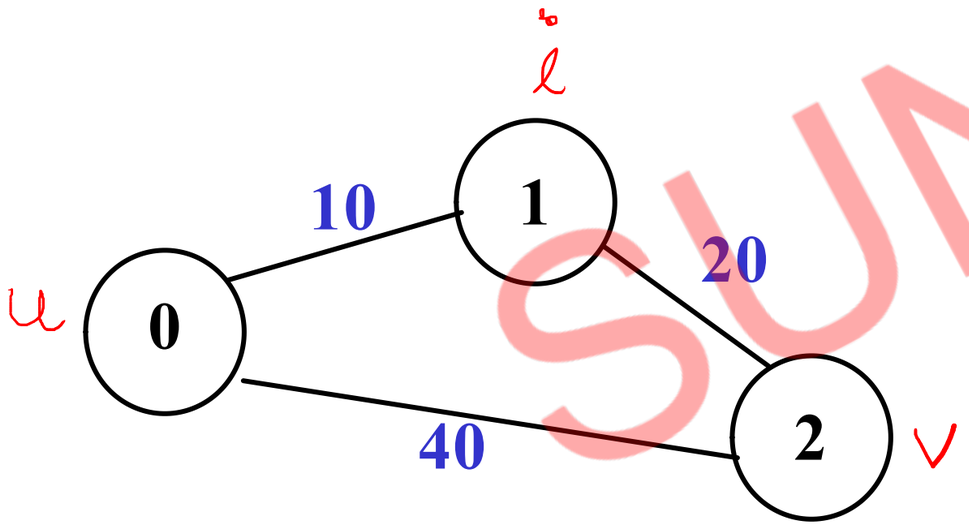
	D
0	4
1	6
2	0
3	5
4	12
5	5
6	9

SUNBEAM

Floyd Warshall Algorithm

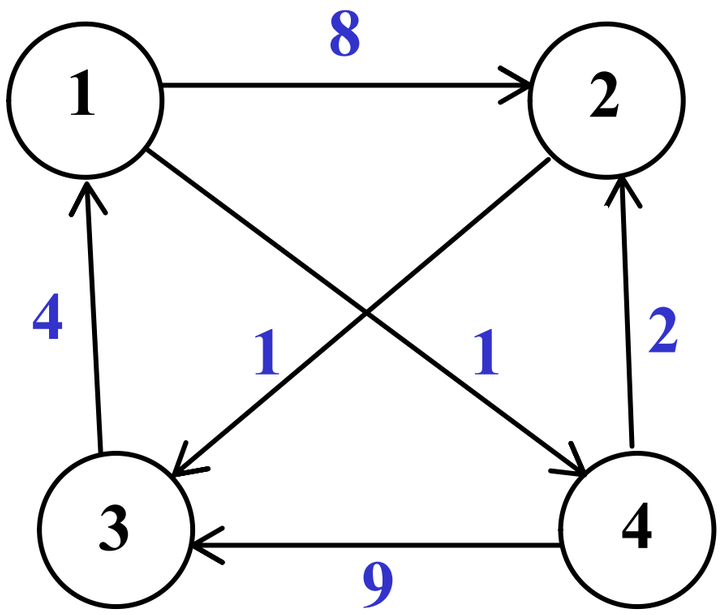
1. Create distance matrix to keep distance of every vertex from each vertex.
   Initially assign it with weights of all edges among vertices
   (i.e. adjacency matrix).

2. Consider each vertex (i) in between pair of any two vertices (s, d) and
   find the optimal distance between s & d considering intermediate vertex
   i.e. dist(s,d) = dist(s,i) + dist(i,d),
   if dist(s,i) + dist(i,d) < dist(s,d).



if (dist[0-1] + dist[1-2] < dist[0-2])
dist[0-2] = dist[0-1] + dist[1-2]

SUNBEAM



	1	2	3	4
1	∞	8	∞	1
2	∞	∞	1	∞
3	4	∞	∞	∞
4	∞	2	9	∞

$A^0 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$A^1 =$

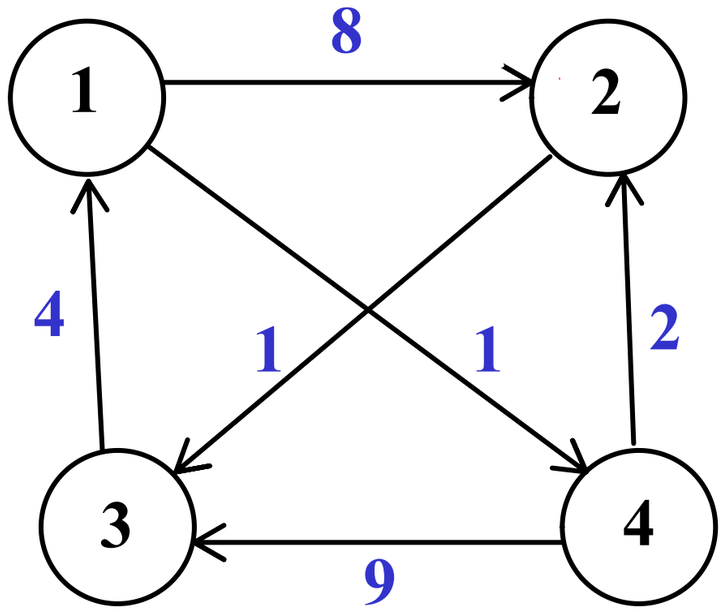
	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

4-3 = 9

4-2+2-3

2+1 = 3

SUNBEAM



$A^2 =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

$A^3 =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

$A^4 =$

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

SUNBEAM

Floyd Warshall Algorithm

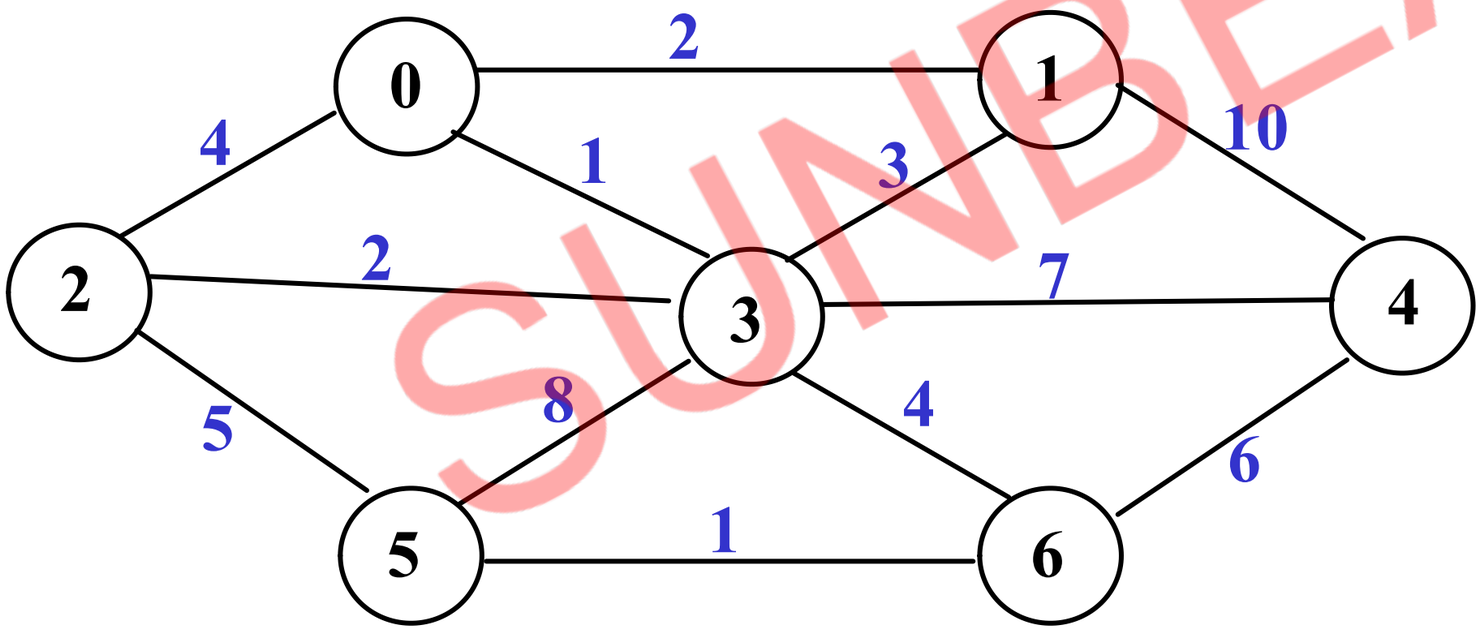
```
int dist[][] = new int[vCount][vCount];
for(int u = 0 ; u < vCount ; u++){
        for(int v = 0 ; v < vCount ; v++){
                dist[u][v] = adjmat[u][v];
        }
        dist[u][u] = 0;
}

for(int i = 0 ; i < vCount ; i++){
        for(int u = 0 ; u < vCount ; u++){
                for(int v = 0 ; v < vCount ; v++){
                        if(dist[u][i] + dist[i][v] < dist[u][v])
                                dist[u][v] = dist[u][i] + dist[i][v];
                }
        }
}
```

SUNBEAM

Union Find Algorithm

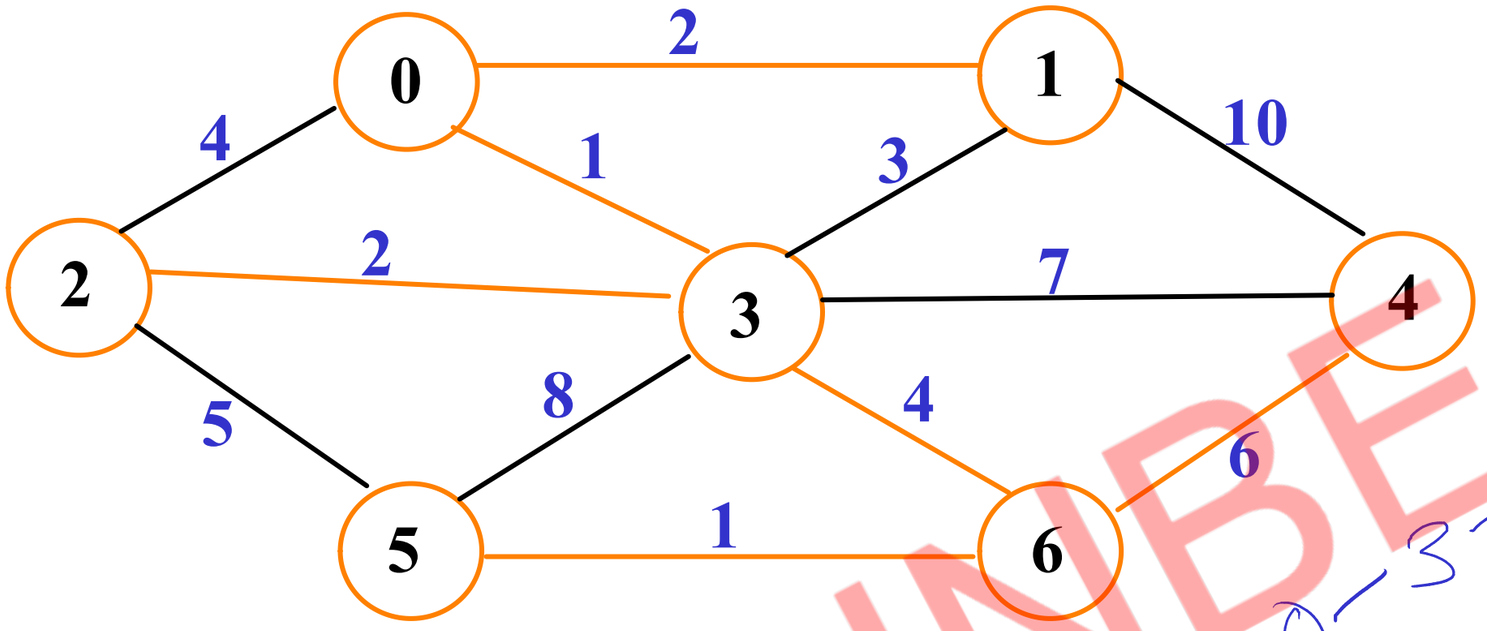
1. Consider all vertices as disjoint sets (parent = -1).
2. For each edge in the graph
    1. Find set(root) of first vertex.
    2. Find set(root) of second vertex.
    3. If both are in same set(same root), cycle is detected.
    4. Otherwise, merge(Union) both the sets i.e. add root of first set under second set



SUNBEAM

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

Parent:



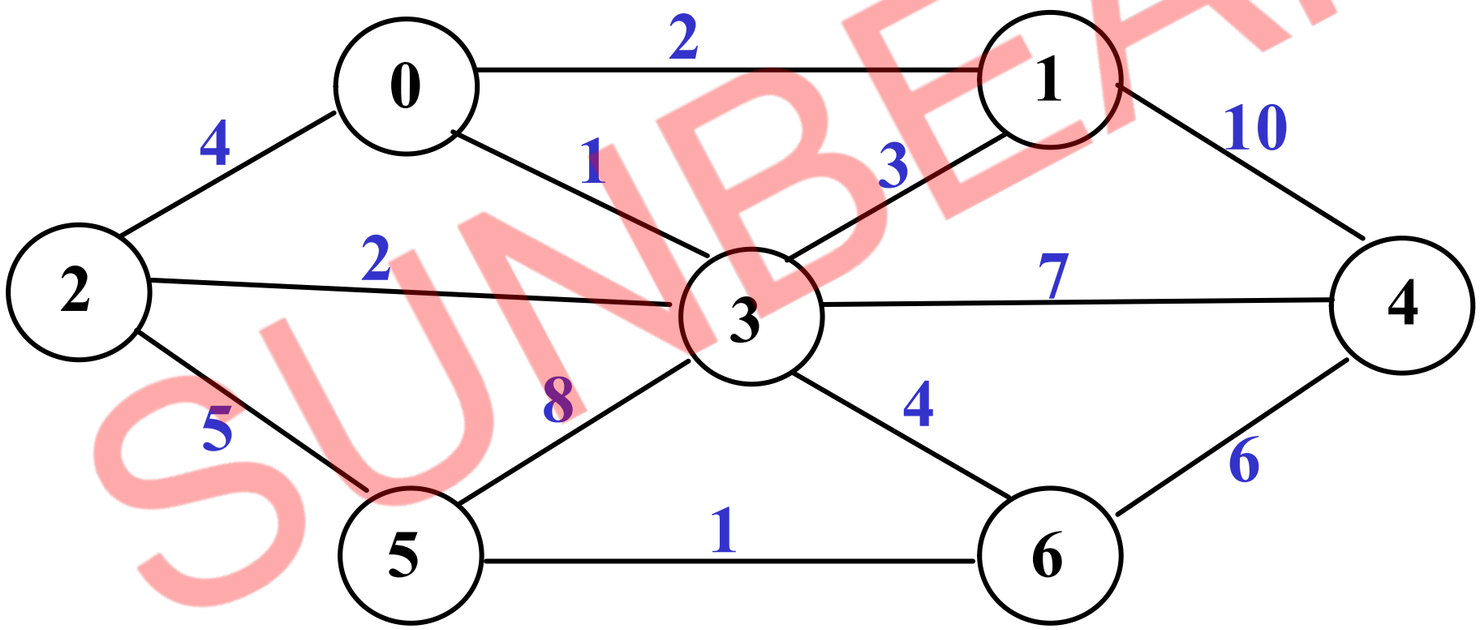

```
int find(int V, int parent[])
    while(parent[V] != -1)
        V=parent[V];
    return V;
}
```

```
void union(int sr, int dr, int parent[])
{
    parent[sr] = dr;
}
```

SUNBEAM

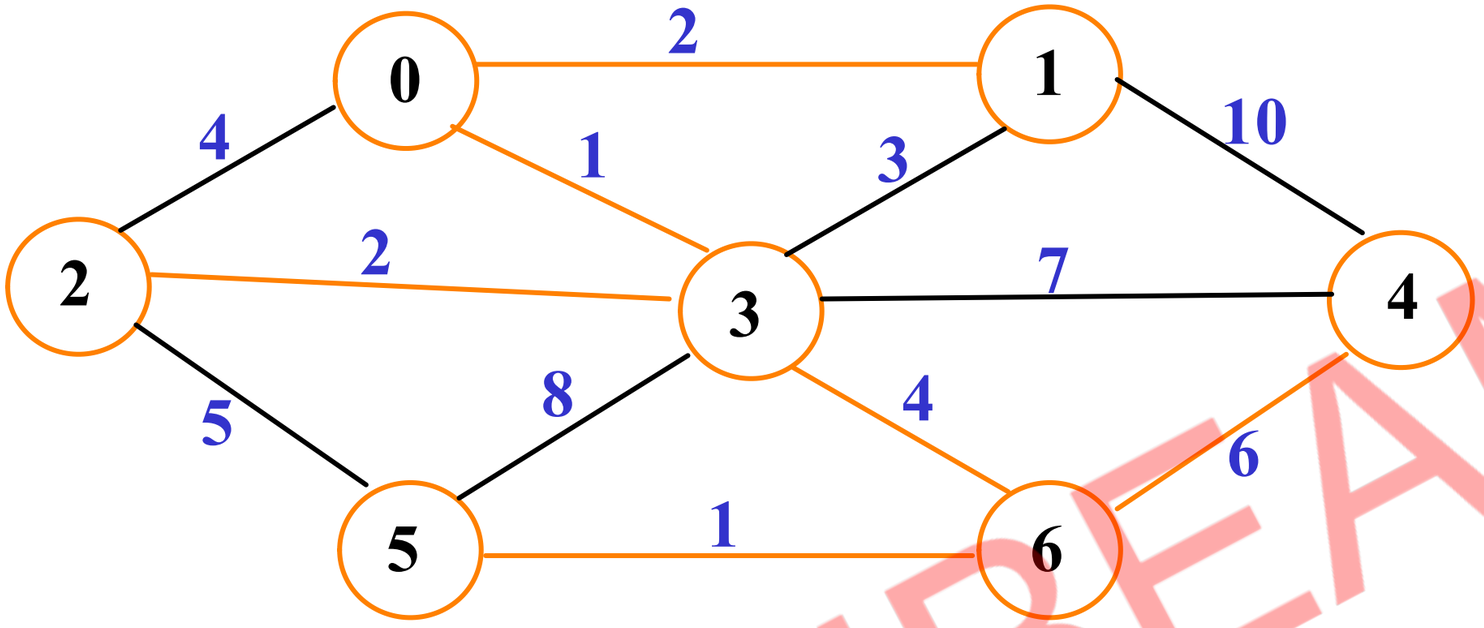
Kruskal's MST

1. Sort all the edges in ascending order of their weight.
2. Pick the smallest edge.
   Check if it forms a cycle with the spanning tree formed so far.
   If cycle is not formed, include this edge.
   Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spanning tree.



SUNBEAM

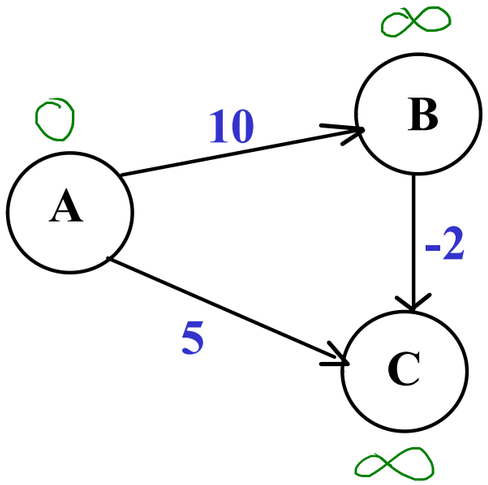
Kruskal's MST

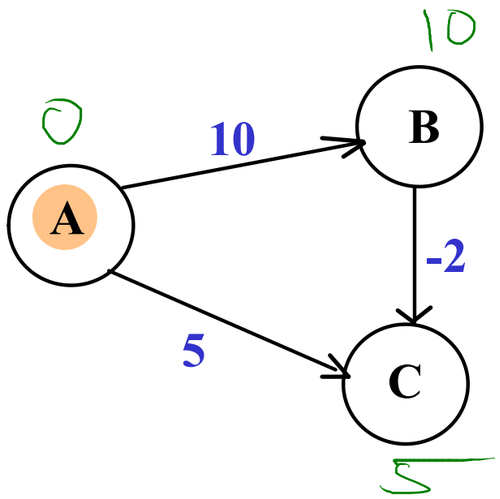


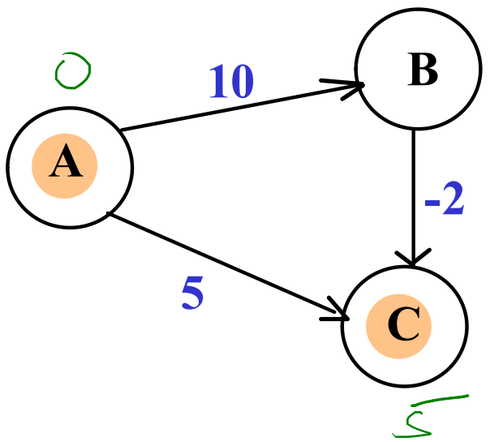
0 3 - 1 ✓
6 5 - 1 ✓
0 1 - 2 ✓
3 2 - 2 ✓
1 3 - 3 ✗
2 0 - 4 ✗
3 6 - 4 ✓
2 5 - 5 ✗
4 6 - 6 ✓
3 4 - 7
3 5 - 8
1 4 - 10

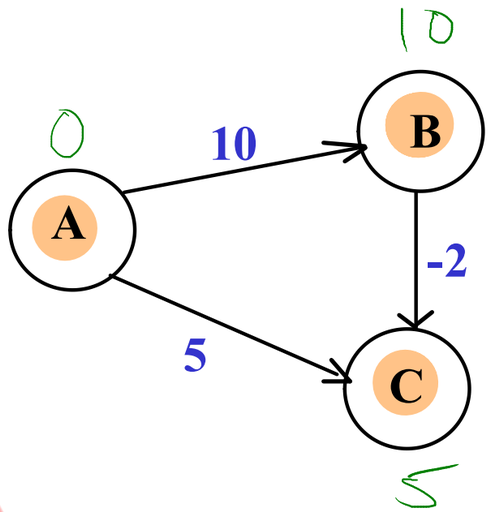
MST weight = 16

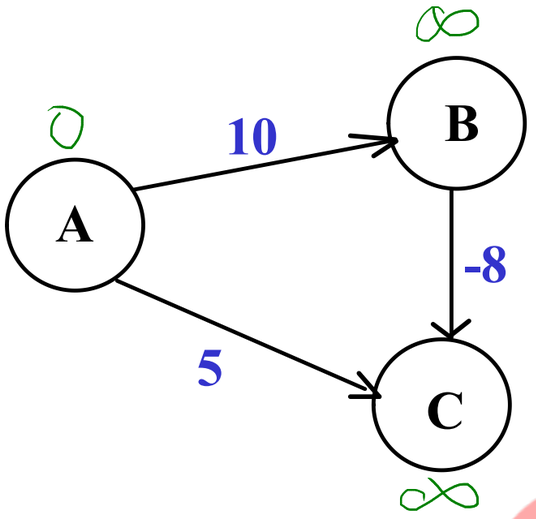
SUNBEAM

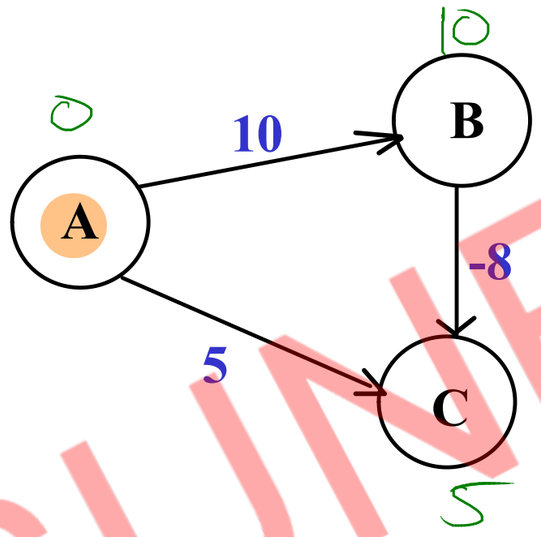

∞
0
10
B
A
-2
5
C
∞

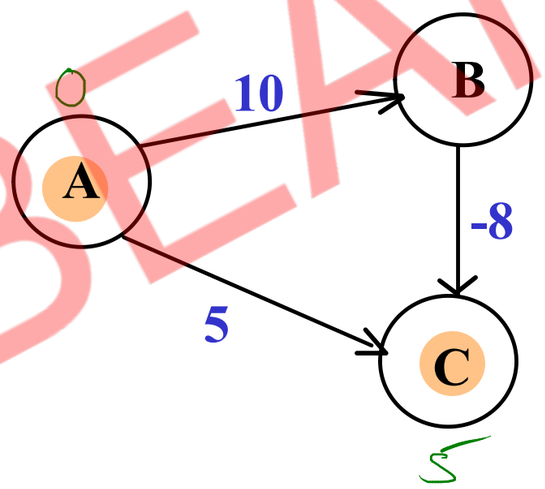

10
0
10
B
A
-2
5
C
5

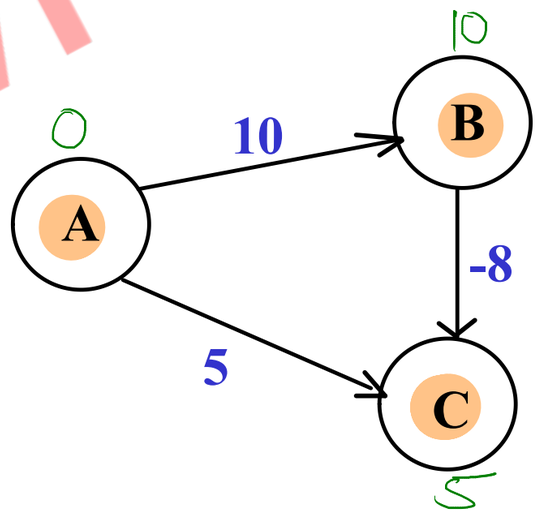

0
10
B
A
-2
5
C
5

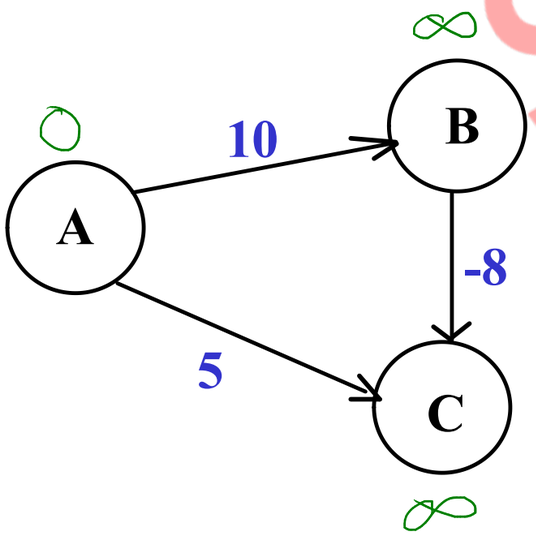

10
0
10
B
A
-2
5
C
5

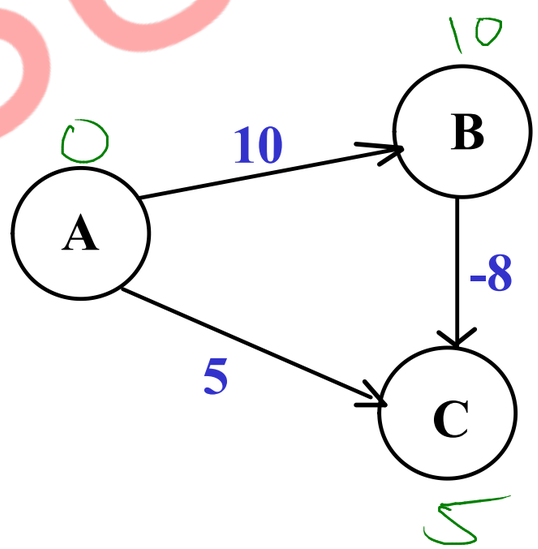

∞
0
10
B
A
-8
5
C
∞

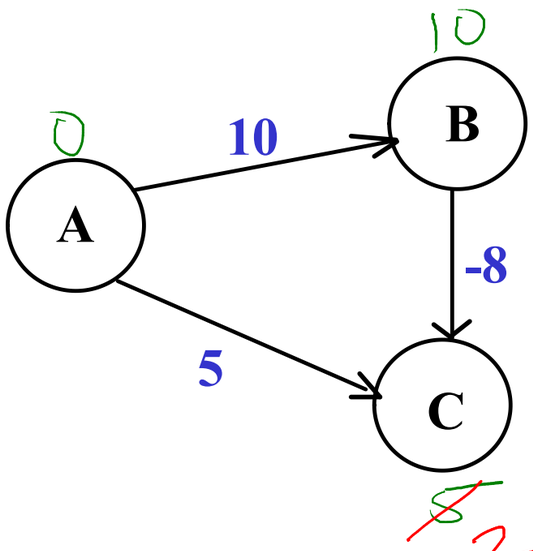

10
0
10
B
A
-8
5
C
5

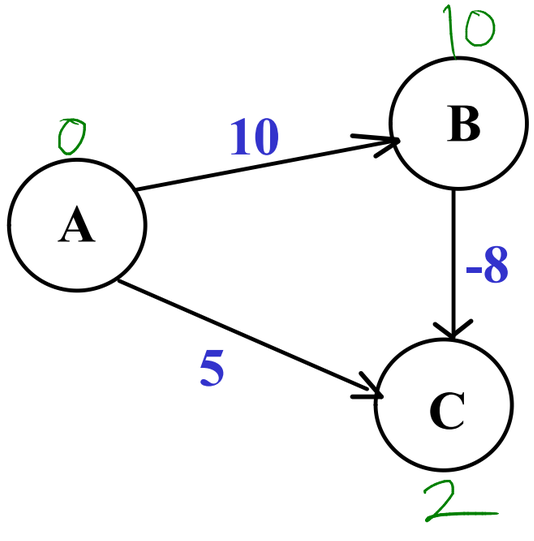

0
10
B
A
-8
5
C
5


10
0
10
B
A
-8
5
C
5


∞
0
10
B
A
-8
5
C
∞


10
0
10
B
A
-8
5
C
5


10
0
10
B
A
-8
5
C
5
2

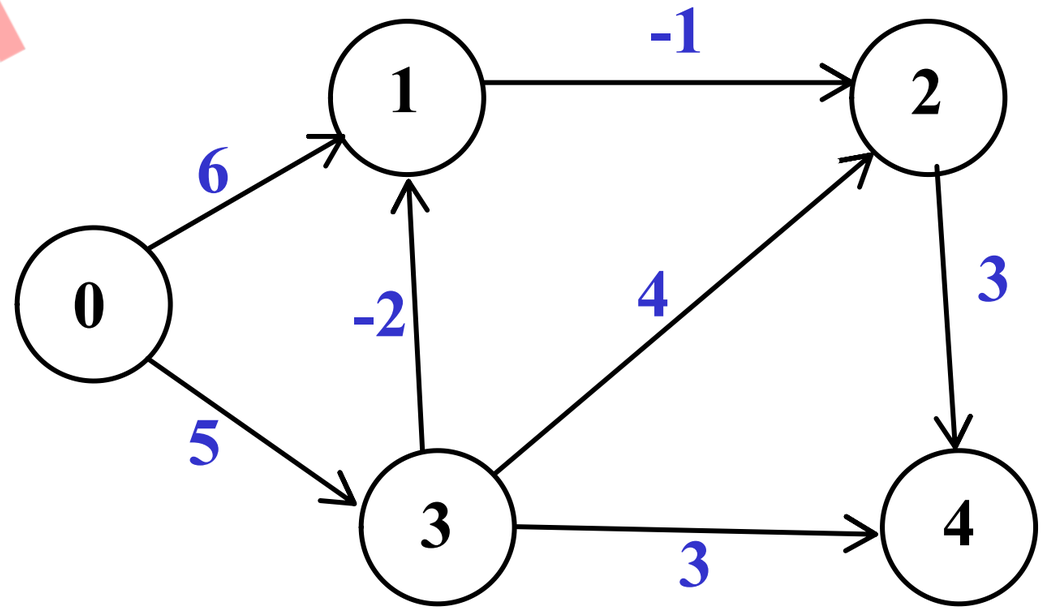

10
0
10
B
A
-8
5
C
2

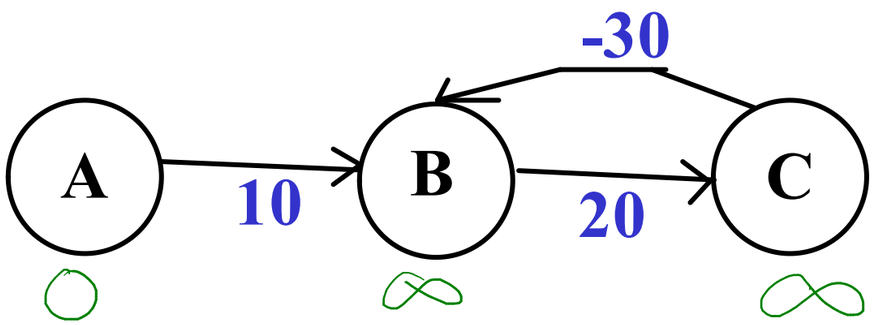
SUNBEAM

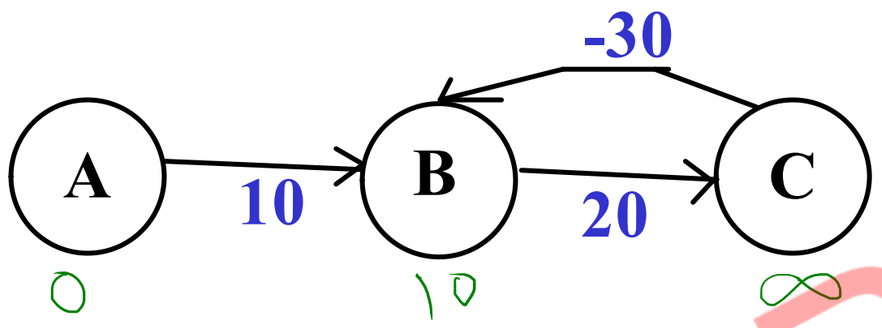
Bellman Ford Algorithm

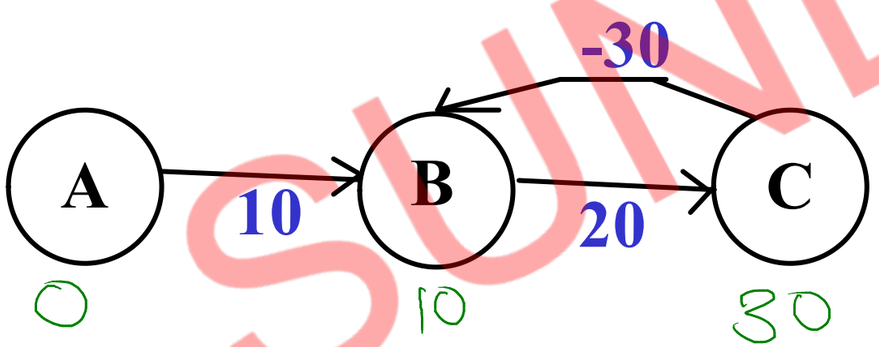
1. Initializes distances from the source to all vertices as infinite and distance to the source itself as 0.
2. Calculates shortest distance V-1 times:
   For each edge u-v,
   if dist[v] > dist[u] + weight of edge u-v,
   then update dist[v], so that
   dist[v] = dist[u] + weight of edge u-v.
3. Check if negative edge cycle in the graph:
   For each edge u-v,
   if dist[v] > dist[u] + weight of edge (u,v),
   then graph has -ve weight cycle.

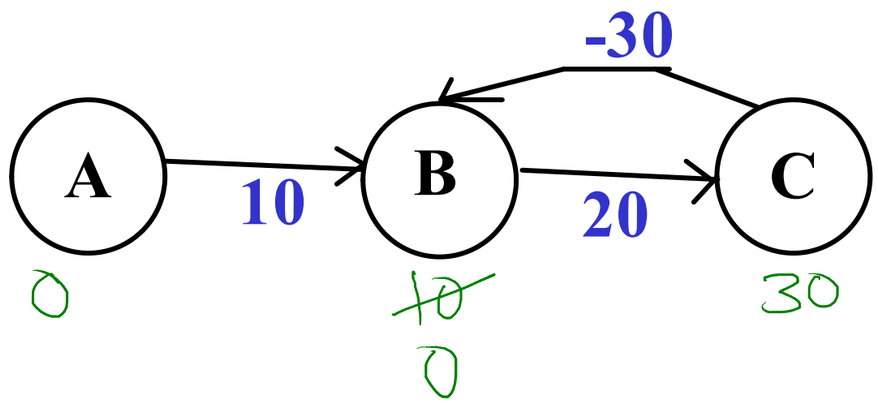
SUNBEAM




-30
A
10
B
20
C
0
∞
∞


-30
A
10
B
20
C
0
١٥
∞


-30
A
10
B
20
C
0
10
30


-30
A
10
B
20
C
0
10
0
30

SUNBEAM

Graph applications

- Graph represents flow of computation/tasks. It is used for resource planning and scheduling. MST algorithms are used for resource conservation. DAG are used for scheduling in Spark or Tez.

- In OS, process and resources are treated as vertices and their usage is treated as edges. This resource allocation algorithm is used to detect deadlock.

- In social networking sites, each person is a vertex and their connection is an edge. In Facebook person search or friend suggestion algorithms use graph concepts.

- In world wide web, web pages are like vertices; while links represents edges. This concept can be used at multiple places.
    - Making sitemap
    - Downloading website or resources
    - Developing web crawlers
    - Google page-rank algorithm

- Maps uses graphs for showing routes and finding shortest paths. Intersection of two (or more) roads is considered as vertex and the road connecting two vertices is considered to be an edge.

SUNBEAM

Merge Sort

//1. divide array into two parts
//2. sort two partitions individually (by applying same merge sort algorithm)
//3. merge sorted partitions into one temporary array
//4. overwrite temporary array into original array

No. of elements = n

levels of division = $\log n$

No. of comparisions per level $\propto n$

Total comps = $n * \log n$

Time $\propto n \log n$

Best, Avg, Worst

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

Processing variables → temp array

Auxillary space $\propto n$

$$S(n) = O(n)$$

SUNBEAM

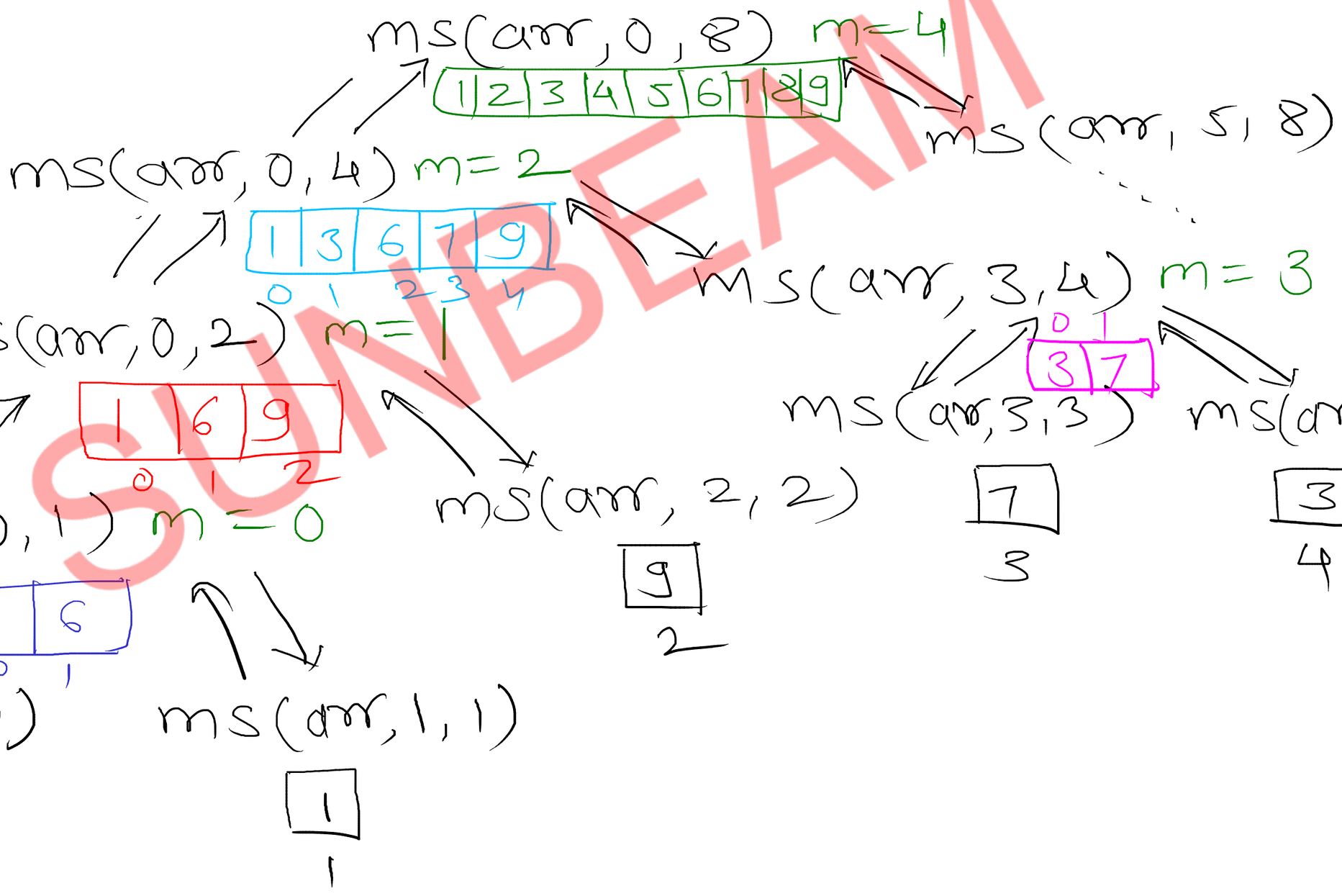
1 2 3 4 5 6 7 8 9

1 3 6 7 9

1 6 8 3 7 2 4 5 8

1 6 2 8 4 8

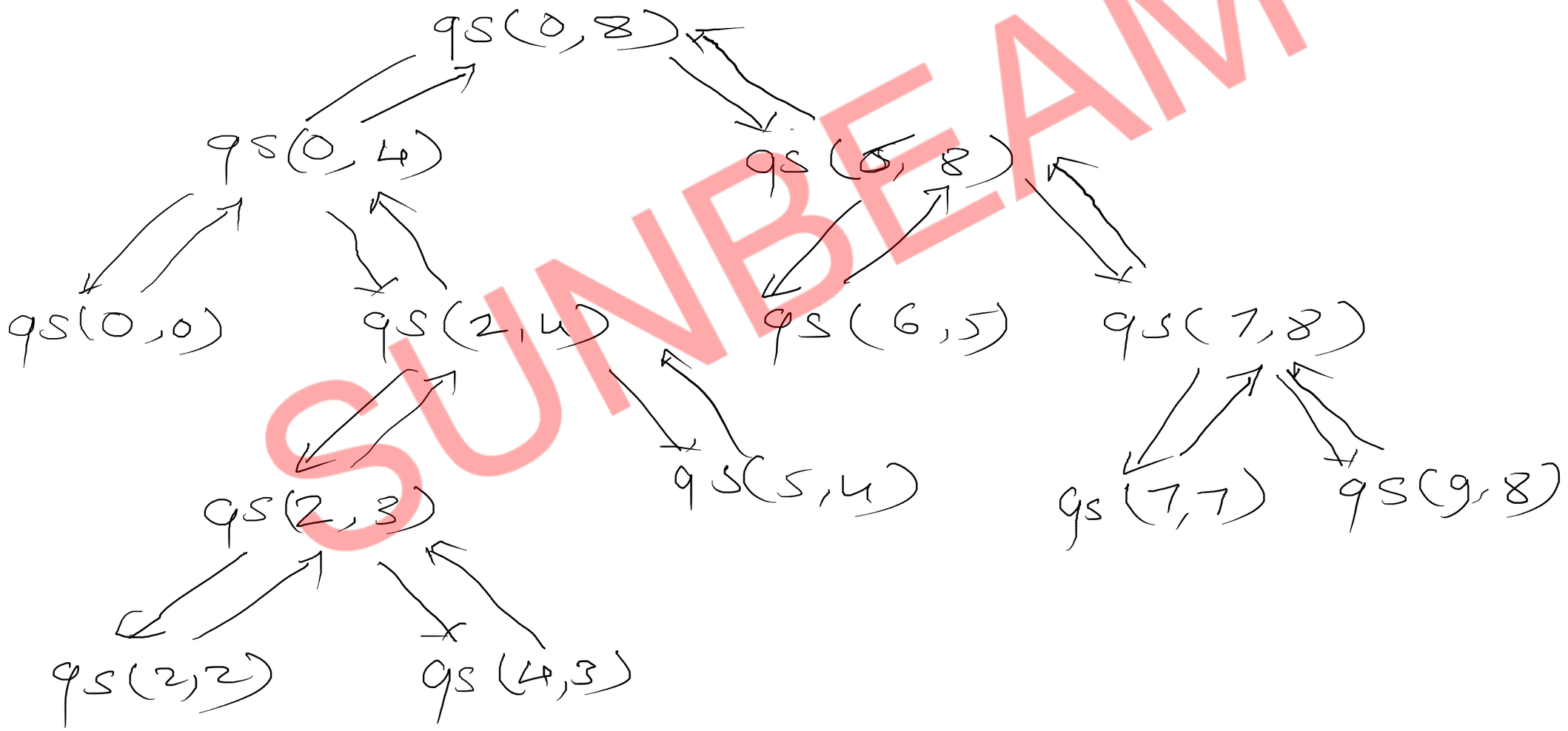
6	1	9	7	3	8	2	4	5
0	1	2	3	4	5	6	7	8



SUNBEAM

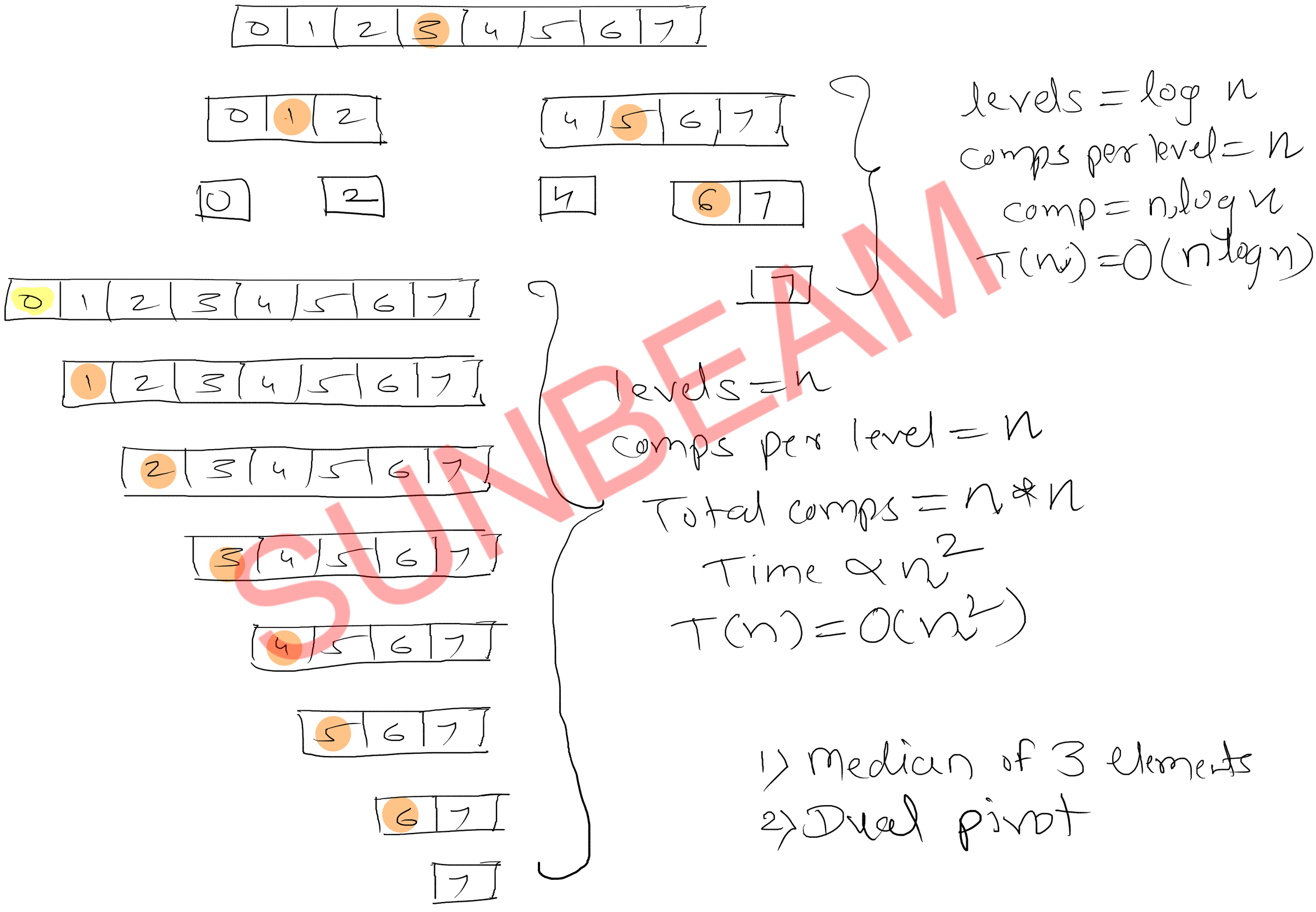
Quick Sort

//1. select pivot/axis/reference element from array
//2. arrange smaller elements than pivot on left side it
//3. arrange greater elements than pivot on right side it
//4. sort left and right partitions of pivot individually by same algorithm



SUNBEAM

Quick Sort



SUNBEAM

Algorithm Design Techniques

1. Divide and Conquor

- divide bigger problem into small problems and solve them seperately

- conquor(merge) solution of sub problems to get final solution

e.g. quick sort and merge sort

2. Greedy

- take local decisions according to current situation and go ahead

- never re think about the decisions which we have taken in past

e.g. primsMST, djkhstraSPT, KruskalsMST

3. Dynamic Programming

e.g. Floyd Warshal, Bellaman Ford

Recursion

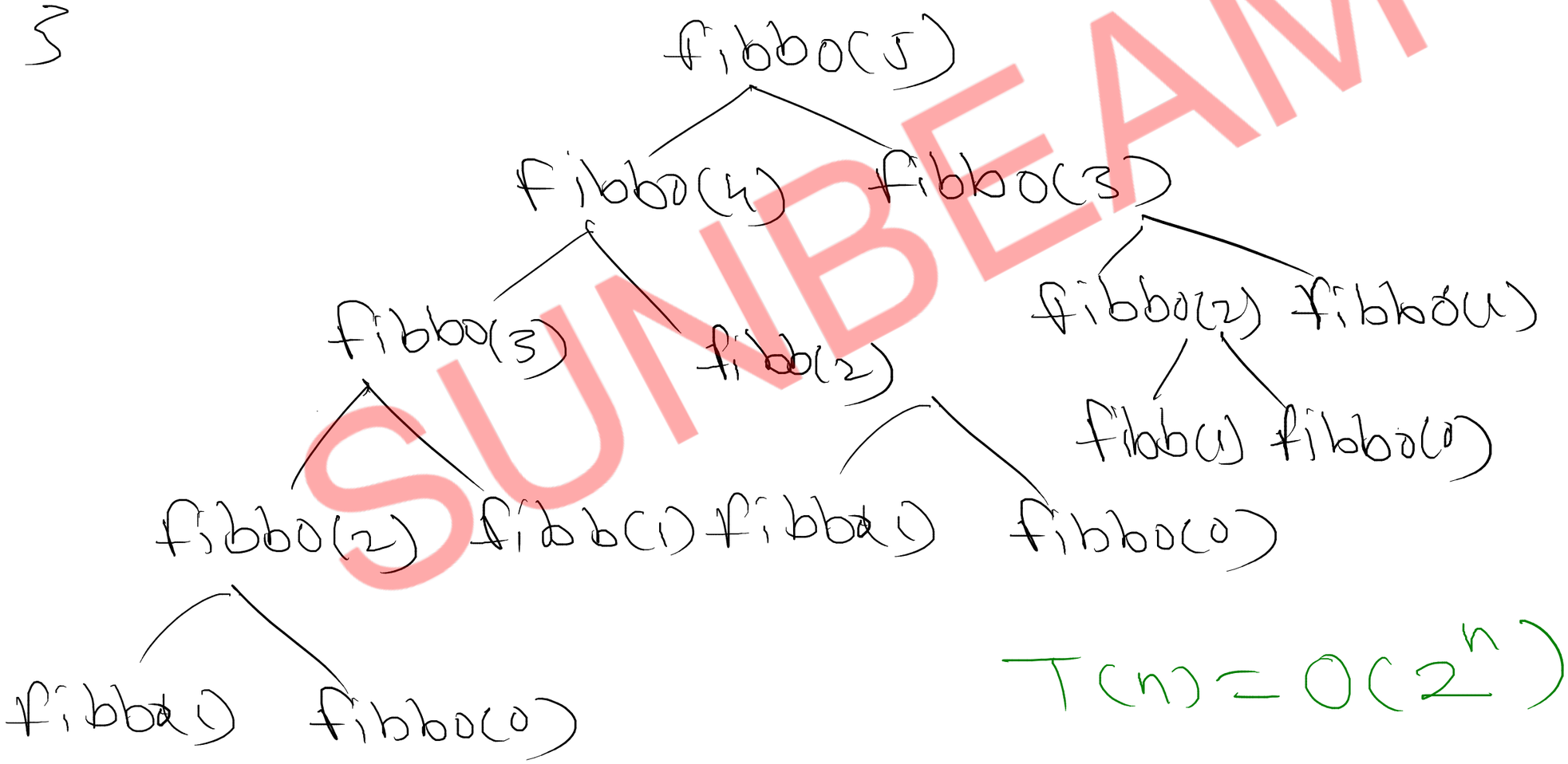
Memoisation
recursion
+
1D or 2D arrays
(bottom up
approach)

Tabulation
loops
+
1D or 2D arrays
(Top down approach)

SUNBEAM

Recursion

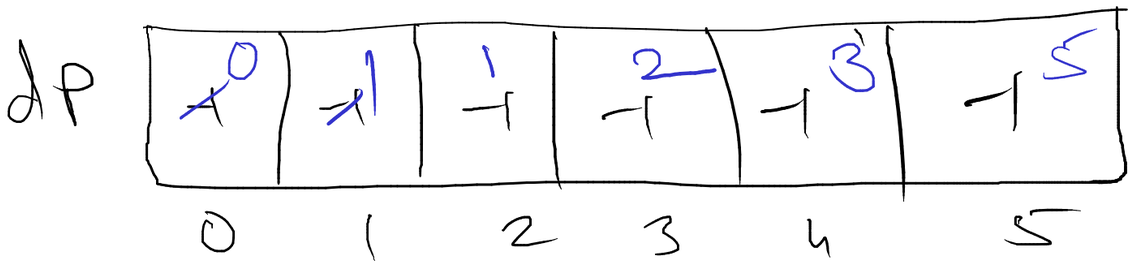
```
int fibbo(int n){
    if(n==0 || n==1)
        return n;
    return fibbo(n-1) + fibbo(n-2);
}
```

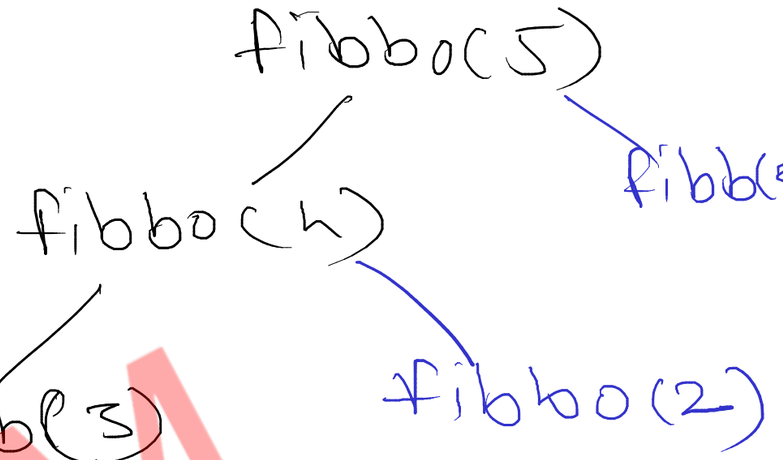


$T(n) = O(2^n)$

SUNBEAM

Memoisation

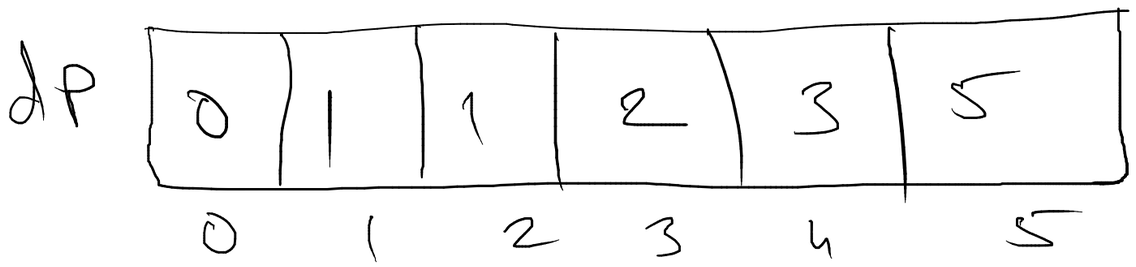





```
int fibbo(int n) {
    if(n==0 || n==1)
        retur n;
    if(dp[n] != -1)
        return dp[n];
    dp[n]=fibbo(n-1)+fibbo(n-2)
    return dp[n];
}
```

SUNBEAM

Tabulation




```
int fibbo(int n) {
    dp[0] = 0;
    dp[1] = 1;
    for(i=2; i<=n; i++) {
        dp[i] = dp[i-1] + dp[i-2];
    }
    return dp[n];
}
```

SUNBEAM

Problem solving technique: Greedy approach

- A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
- We can make choice that seems best at the moment and then solve the sub-problems that arise later.
- The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub-problem.
- It iteratively makes one greedy choice after another, reducing each given problem into a smaller one.
- A greedy algorithm never reconsiders its choices.
- A greedy strategy may not always produce an optimal solution.

eg. Greedy algorithm decides minimum number of coins to give while making change.
coins available : 50, 20, 10, 5, 2, 1

SUNBEAM

Recursion

- Function calling itself is called as recursive function.
- For each function call stack frame is created on the stack.
- Thus it needs more space as well as more time for execution.
- However recursive functions are easy to program.
- Typical divide and conquer problems are solved using recursion.
- For recursive functions two things are must
- Recursive call (Explain process it terms of itself)
- Terminating or base condition (Where to stop)

eg. Fibonacci Series

- Recursive formula
  - Tn = Tn-1 + Tn-2
- Terminating condition
  - T1 = T2 = 1
- Overlapping sub-problem

SUNBEAM

Memoization

- It's based on the Latin word memorandum, meaning "to be remembered".
- Memoization is a technique used in computing to speed up programs.
- This is accomplished by memorizing the calculation results of processed input such as the results of function calls.
- If the same input or a function call with the same parameters is used, the previously stored results can be used again and unnecessary calculation are avoided.
- Need to rewrite recursive algorithm. Using simple arrays or map/dictionary.

SUNBEAM

Dynamic Programming

- Dynamic programming is another optimization over recursion.
- Typical DP problem give choices (to select from) and ask for optimal result (maximum or minimum).
- Technically it can be used for the problems having two properties
  - Overlapping sub-problems
  - Optimal sub-structure
- To solve problem, we need to solve its sub-problems multiple times.
- Optimal solution of problem can be obtained using optimal solutions of its sub-problems.

- Greedy algorithms pick optimal solution to local problem and never reconsider the choice done.
- DP algorithms solve the sub-problem in a iteration and improves upon it in subsequent iterations.

SUNBEAM