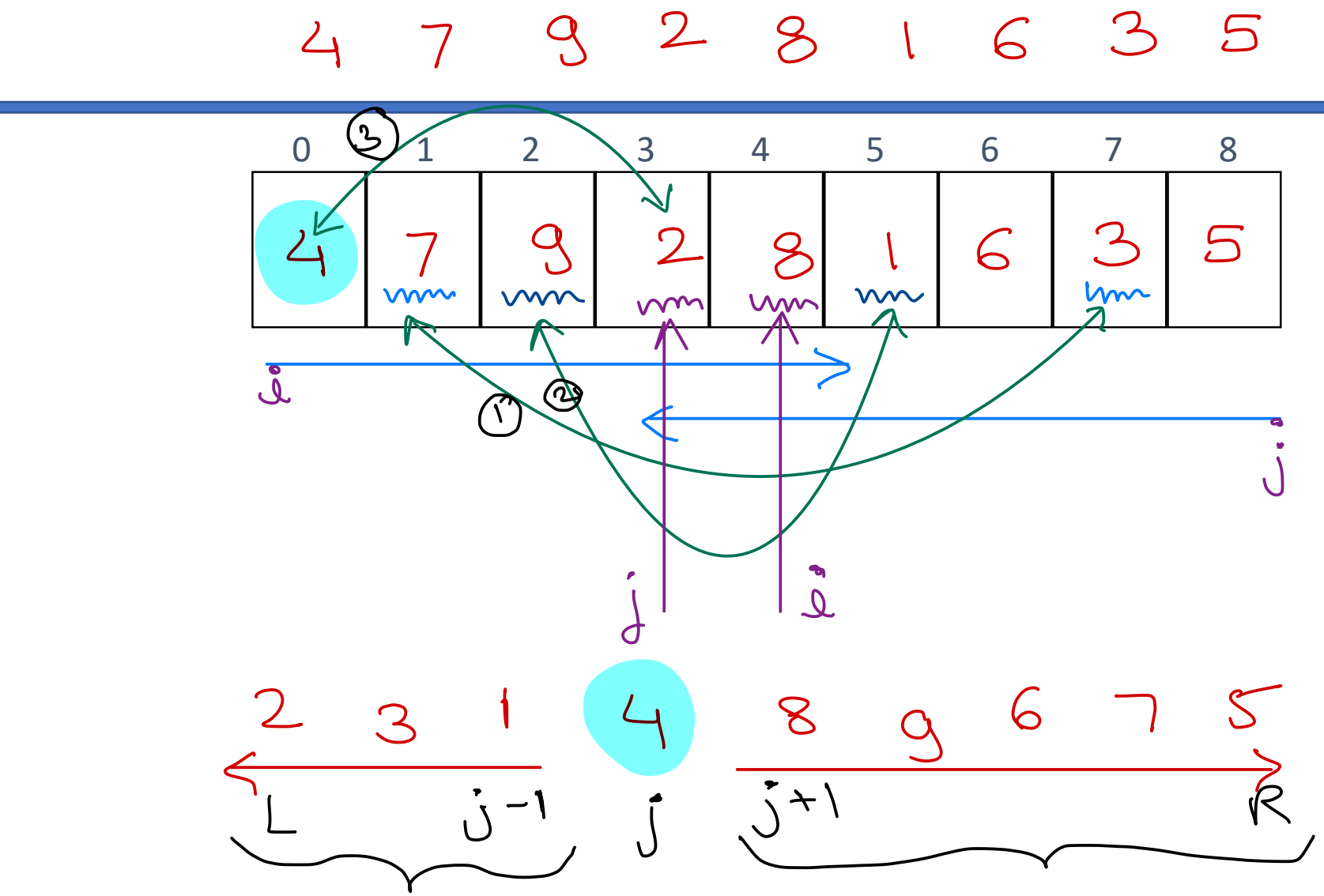
Data Structure & Algorithms

Nilesh Ghule

Quick Sort

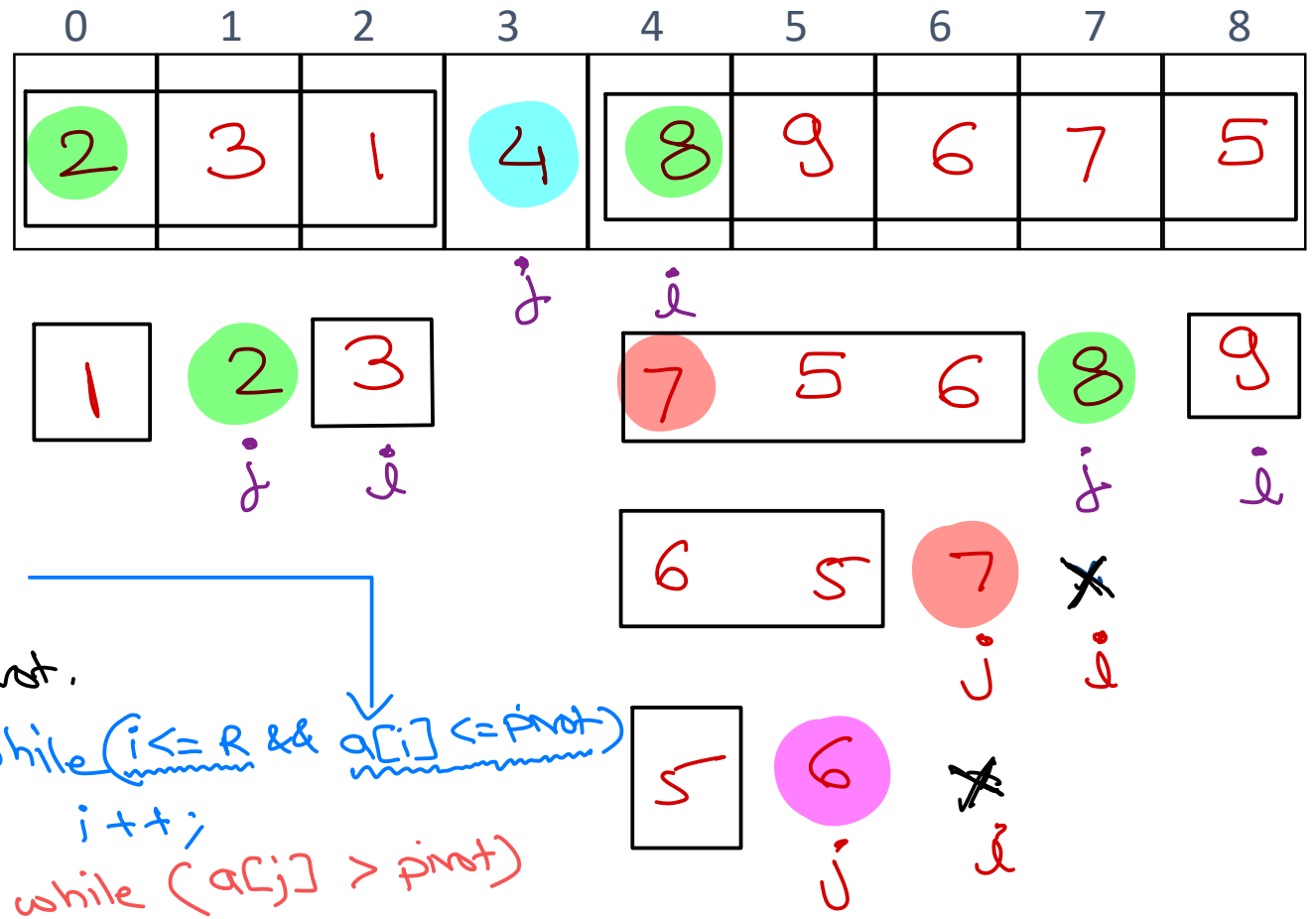


Quick Sort

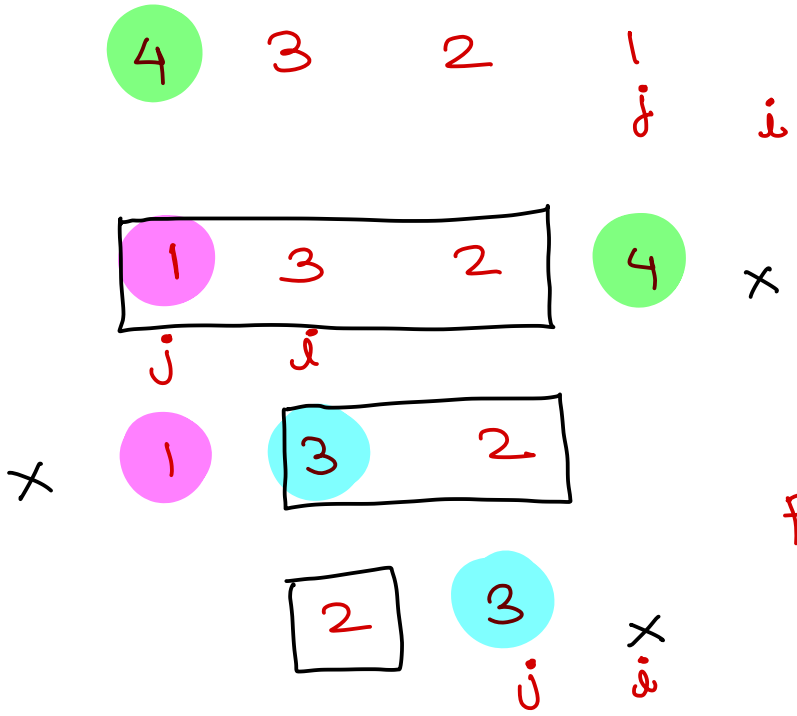
4 7 9 2 8 1 6 3 5

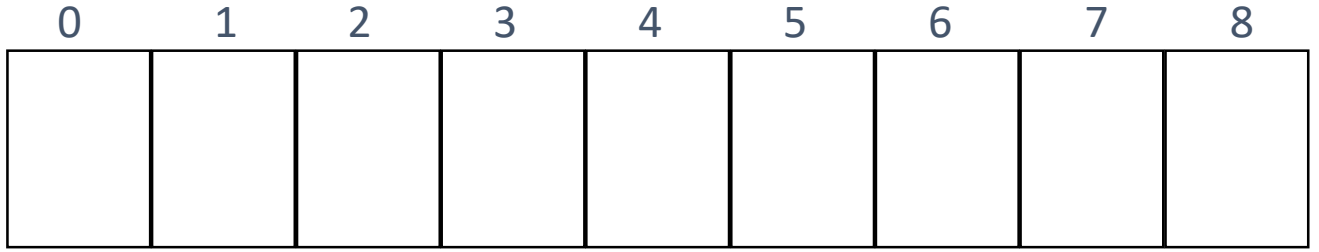
pivot can be ① left most
② right most
③ median

```
quicksort (a, L, R):
    if (L >= R):
        return;
    pivot = a[L];
    i = L, j = R;
    while (i < j){
        from left find (i) ele > pivot.
        from right find (j) ele <= pivot.
        if (i < j)
            swap(a[i], a[j])
    }
    swap (a[j], pivot)
    quicksort (a, L, j-1);
    quicksort (a, j+1, R);
```



Quick Sort





Worst case

partitioning : O(n)

Compar : O(n)

$O(n^2)$

avg/best case

partitioning : O(log n)

Compar : O(n)

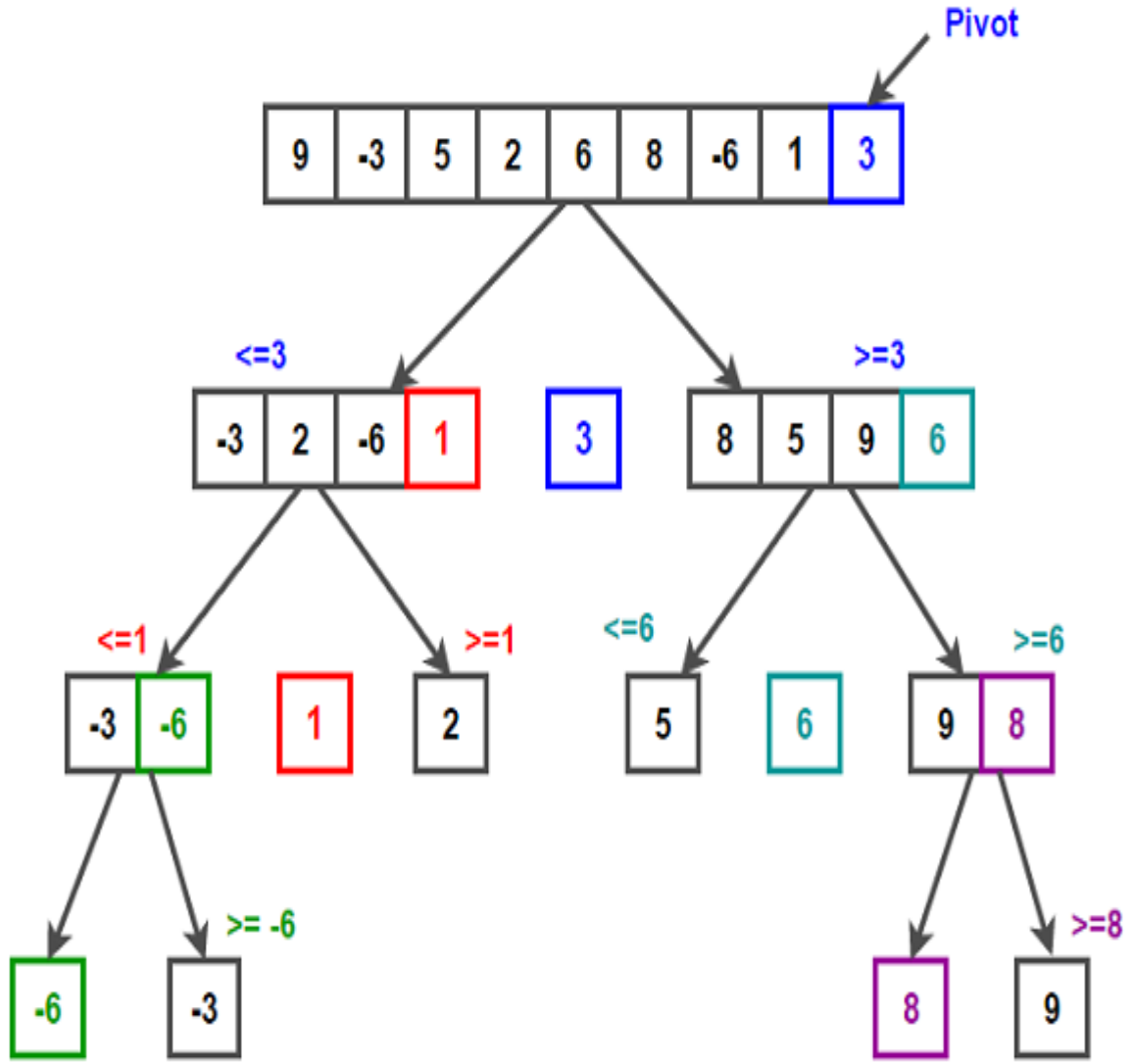
O(n log n)

Quick Sort – Time complexity

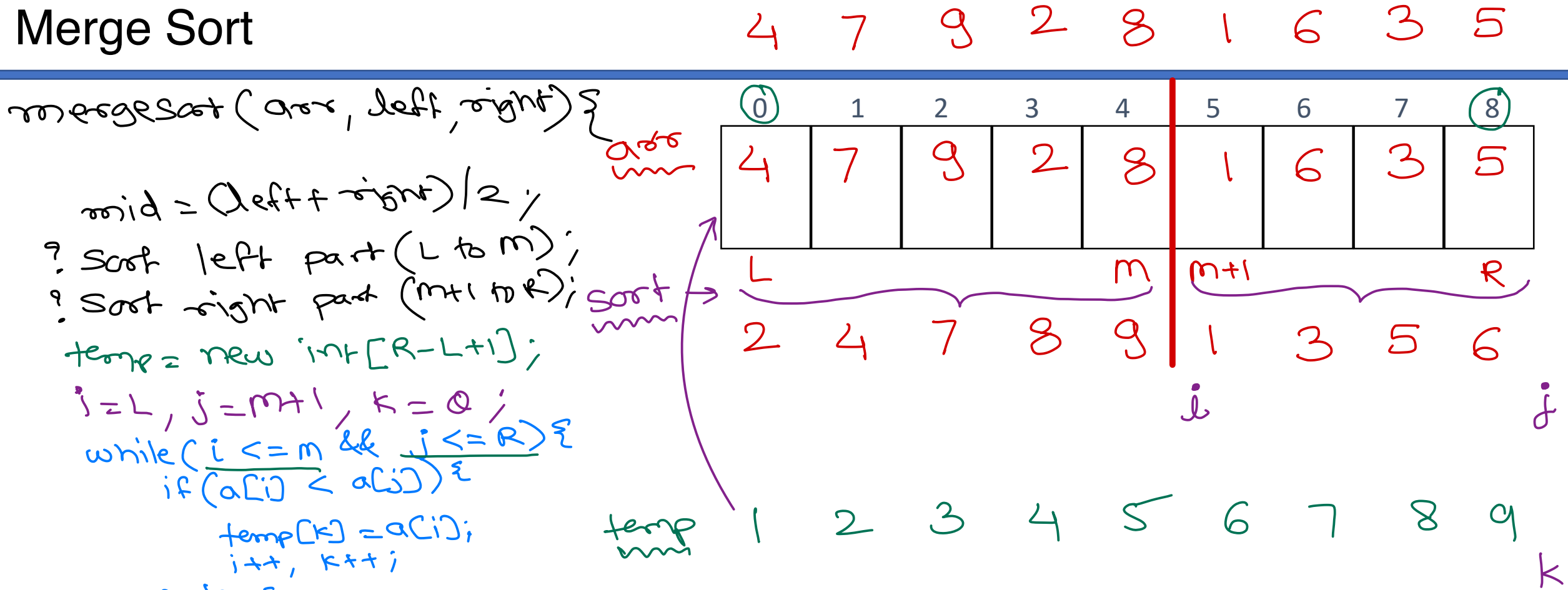
- Quick sort pivot element can be
  - First element or Last element
  - Random element
  - Median of the array
- Quick sort time
  - Time to partition as per pivot – T(n)
  - Time to sort left partition – T(k)
  - Time to sort left partition – T(n-k-1)
- Worst case
  - $T(n) = T(0) + T(n-1) + O(n) \Rightarrow O(n^2)$
- Best case
  - $T(n) = T(n/2) + T(n/2) + O(n) \Rightarrow O(n \log n)$
- Average case
  - $T(n) = T(n/9) + T(9n/10) + O(n) \Rightarrow O(n \log n)$

Recursion – QuickSort

- Algorithm
  1. If single element in partition, return.
  2. Last element as pivot.
  3. From left find element greater than pivot (x^{th} ele).
  4. From right find element less than pivot (y^{th} ele).
  5. Swap x^{th} ele with y^{th} ele.
  6. Repeat 2 to 4 until x < y.
  7. Swap y^{th} ele with pivot.
  8. Apply QuickSort to left partition (left to y-1).
  9. Apply QuickSort to right partition (y+1 to right).
- QS(arr, 0, 8)
  - QS(arr, 0, 3)
    - QS(arr, 0, 1)
      - QS(arr, 0, 0)
    - QS(arr, 3, 3)
  - QS(arr, 5, 8)
    - QS(arr, 5, 5)
    - QS(arr, 7, 8)
      - QS(arr, 9, 9)

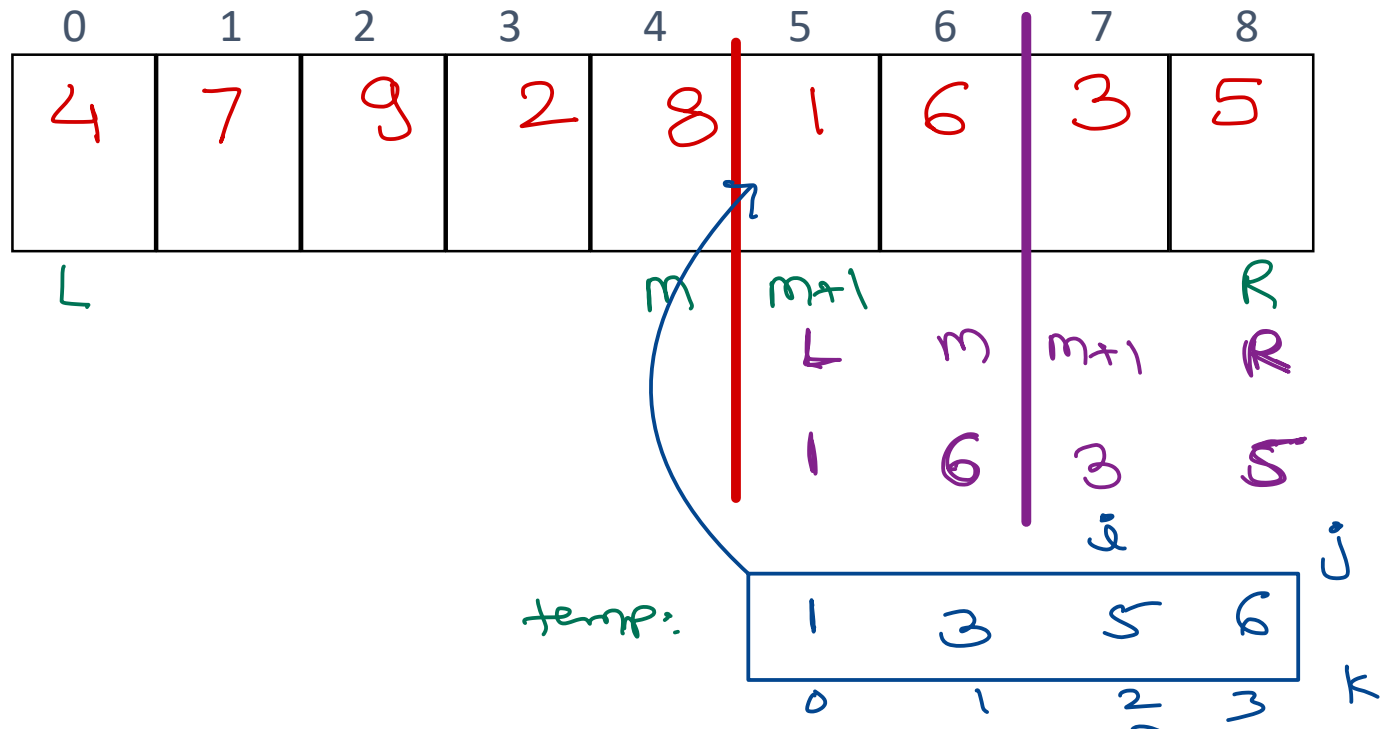


Merge Sort


4 7 9 2 8 1 6 3 5
mergesort(arr, left, right){
mid = (left+right)/2;
? Sort left part (L to m);
? Sort right part (m+1 to R);
temp = new int[R-L+1];
i=L, j=m+1, k=0;
while(i<=m && j<=R){
if(a[i] < a[j]){
temp[k] = a[i];
i++, k++;
}else{
temp[k] = a[j];
j++, k++;
}
while(i<=m){
temp[k] = a[i];
i++, k++;
}
while(j<=R){
temp[k] = a[j];
j++, k++;
}
for(i=0; i<n; i++)
arr[left+i] = temp[i];
arr
0 1 2 3 4 5 6 7 8
4 7 9 2 8 1 6 3 5
L m m+1 R
sort
2 4 7 8 9 1 3 5 6
i j
temp
1 2 3 4 5 6 7 8 9
k

Merge Sort

```
//Sort left part (L to m)
   mergesort(arr, L, m);
// Sort right part (m+1 to R)
   mergesort(arr, m+1, R)
```



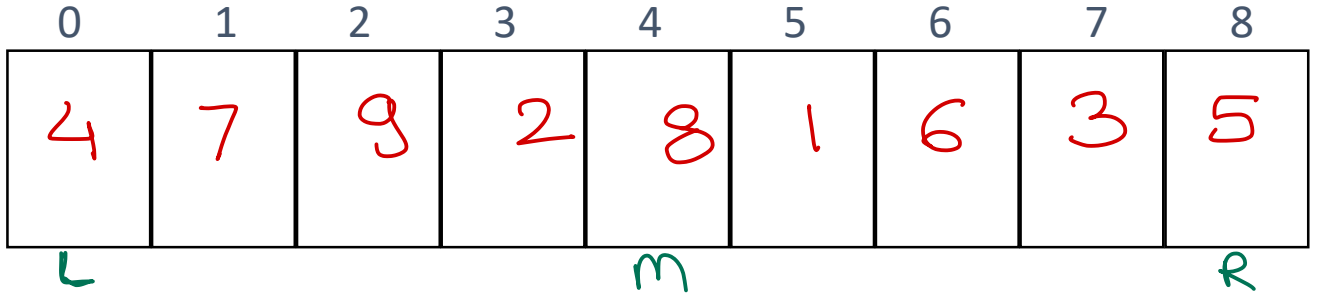

```
for(i=0; i<temp.length ; i++)
    arr[L+i]=temp[i]
```

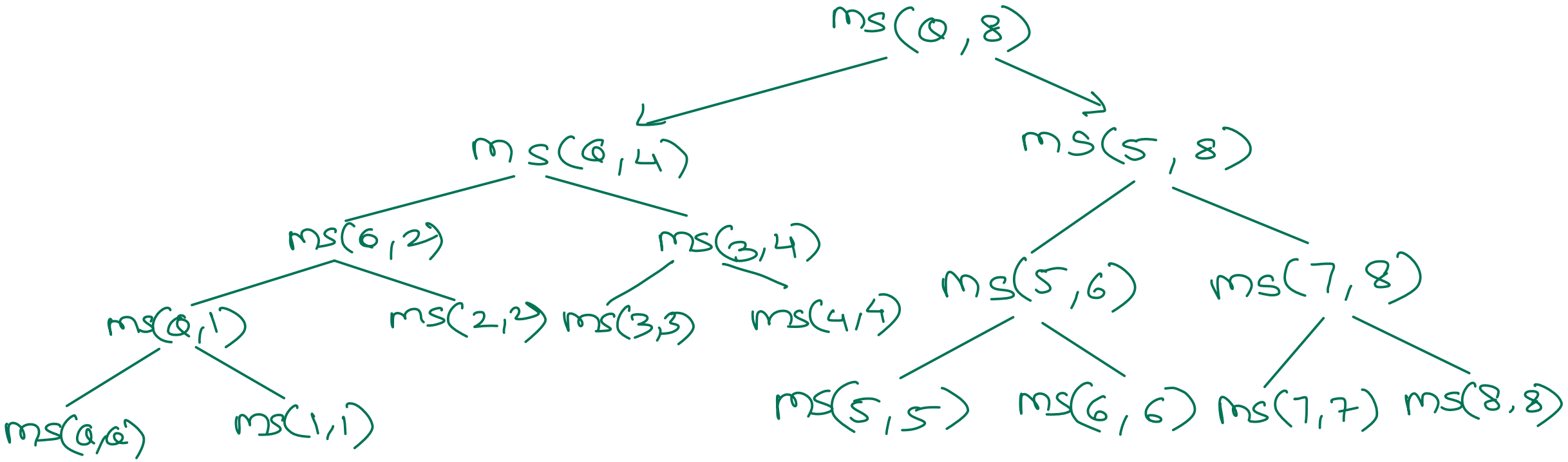
Merge Sort

partitioning → O(log n)

merging → O(n)

merge sort → O(n log n)

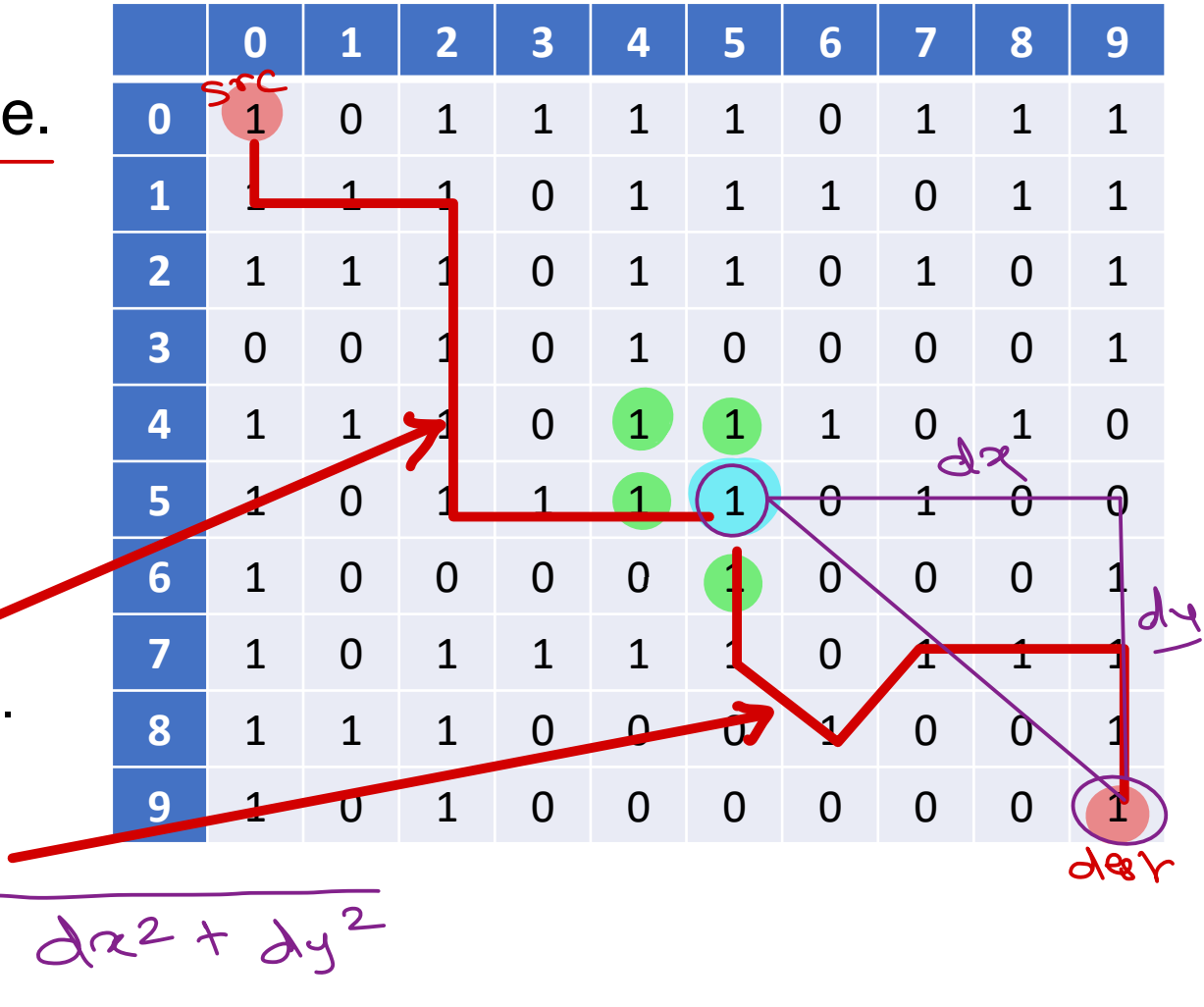




A* Search Algorithm

- Point to point approximate shortest path finder algorithm.
- This algorithm is used in artificial intelligence.
- Commonly used in games or maps to find shortest distance in faster way.
- It is modification of BFS.
- Put selected adjacent vertices on queue, based on some heuristic.
- A math function is calculated for vertices
  - f(v) = g(v) + h(v) → vertex with min f(v) is picked.
  - g(v) → cost of source to vertex v
  - *h(v)* → estimated cost of vertex v to destination.

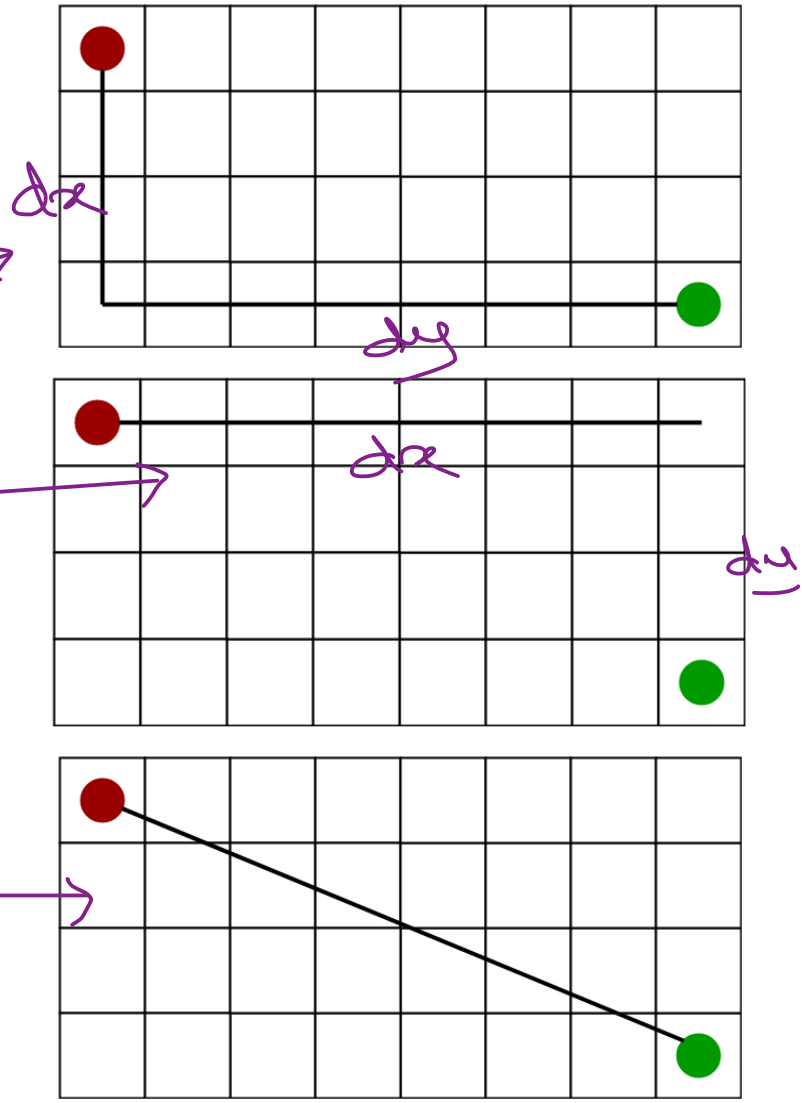
	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	1	0	1	0	0	0	0	1
4	1	1	1	0	1	1	1	0	1	0
5	1	0	1	1	1	1	0	1	0	0
6	1	0	0	0	0	1	0	0	0	1
7	1	0	1	1	1	1	0	1	1	1
8	1	1	1	0	0	0	1	0	0	1
9	1	0	1	0	0	0	0	0	0	1



① $dx + dy$ ② $\min(dx, dy)$ ③ $\sqrt{dx^2 + dy^2}$

A* Search Algorithm

- h(v) represent heuristic and depends on problem domain. Three common techniques to calculate heuristic:
- Manhattan distance
  - When moves are limited in four directions only.
  - h = dx + dy
- Diagonal distance
  - When moves are allowed in all eight directions (one step).
  - h = MAX(dx, dy)
- Euclidean distance
  - When moves are allowed in any direction.
  - h = $\sqrt{dx^2 + dy^2}$
- Note that heuristic may result in longer paths in typical cases.



A* search algorithm

$f(v) = g(v) + h(v)$

$g(v)$ = distance traveled from src

$h(v)$ = distance to travel (heuristic)

- Start point g(v) = 0, h(v) = 0 & f(h) = 0.
- Push start point vertex on a priority queue (by f(v)).
- Until queue is empty
  - Pop a point (v) from queue.
  - Add v into the path.
  - For each adjacent point (u)
    - If u is destination, build the path.
    - If u is invalid or already on path or blocked, skip it.
    - Calculate newg = g(v) + 1, newh = *heuristic* and newf = newg + newh.
    - If newf is less than f(u), f(u) = newf and also parent(u) = v. Rearrange elements in priority queue.

out of maze

0

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

A* Search Algorithm

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

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

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.
  - ↳ Big Data

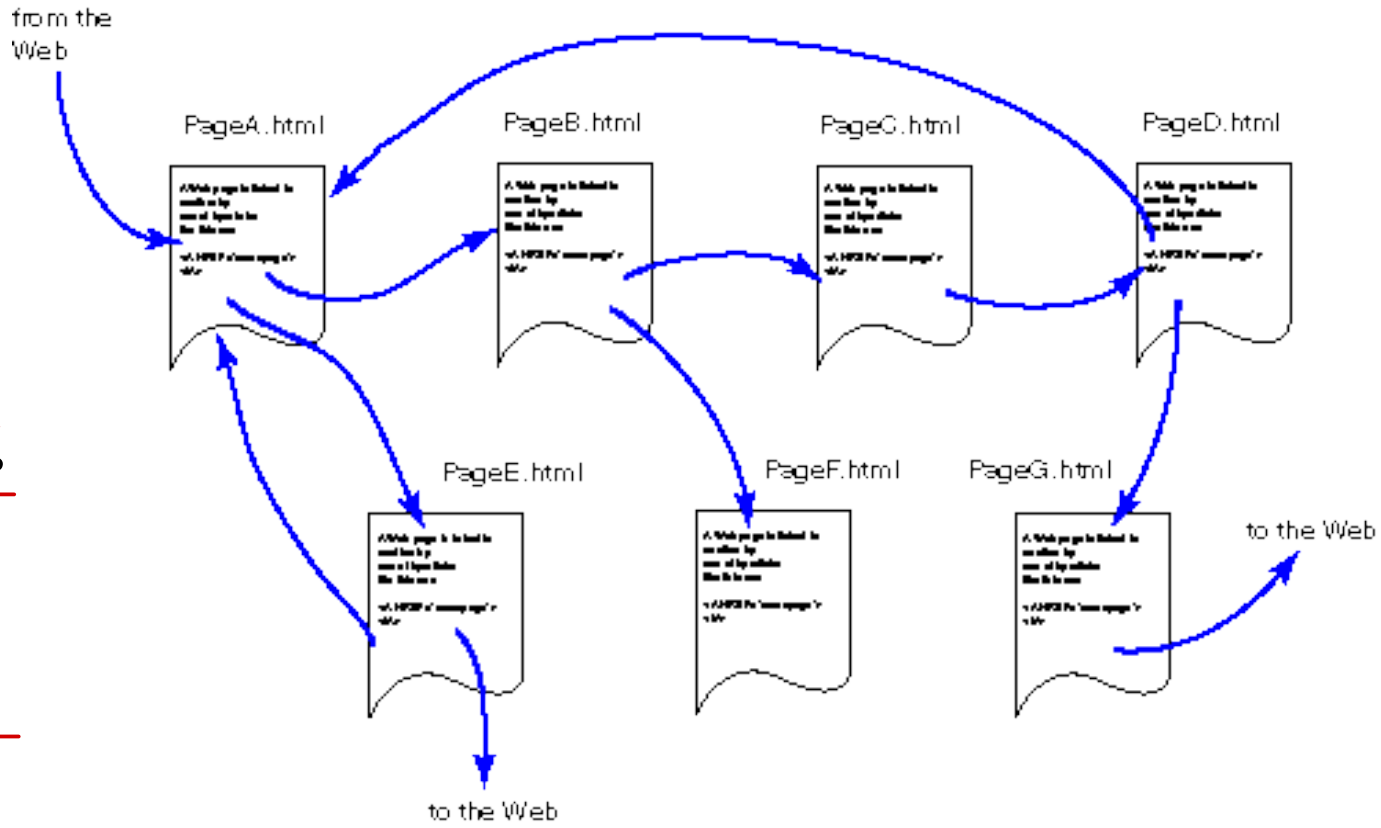
- In OS, process and resources are treated as vertices and their usage is treated as edges. This resource allocation algorithm is used to detect deadlock.

- 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.

Graph applications

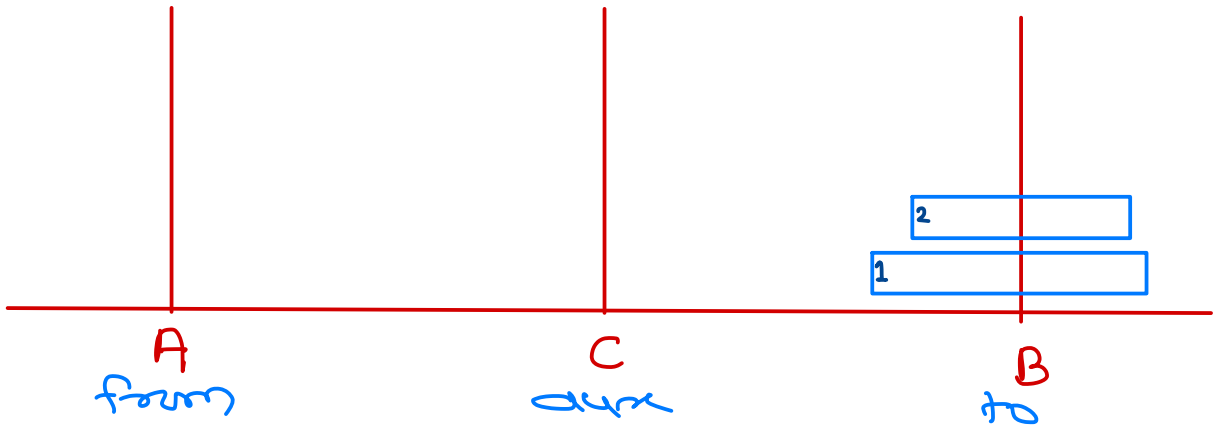
SEO

- 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



PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

Tower of Hanoi



num of disks = 1

① A → B

num of disks = 2

① 2 A → C

② 1 A → B

③ 2 C → B

① transfer all disks from A to B.

② you can use aux rod C.

③ can transfer only one disk at a time.

④ only smaller disk can be kept on larger disk.

from → aux

from → to

aux → to

Graph applications

- 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.

all-pair shortest path

src	1	2	3	4	5	6	7	8	9	10	11
1	0	40	77								
→ 2	40	0	117								
3											
4											
5											
6											
7											
8											
9											
10											
11