Algo Analysis

BFS: O(V+E)

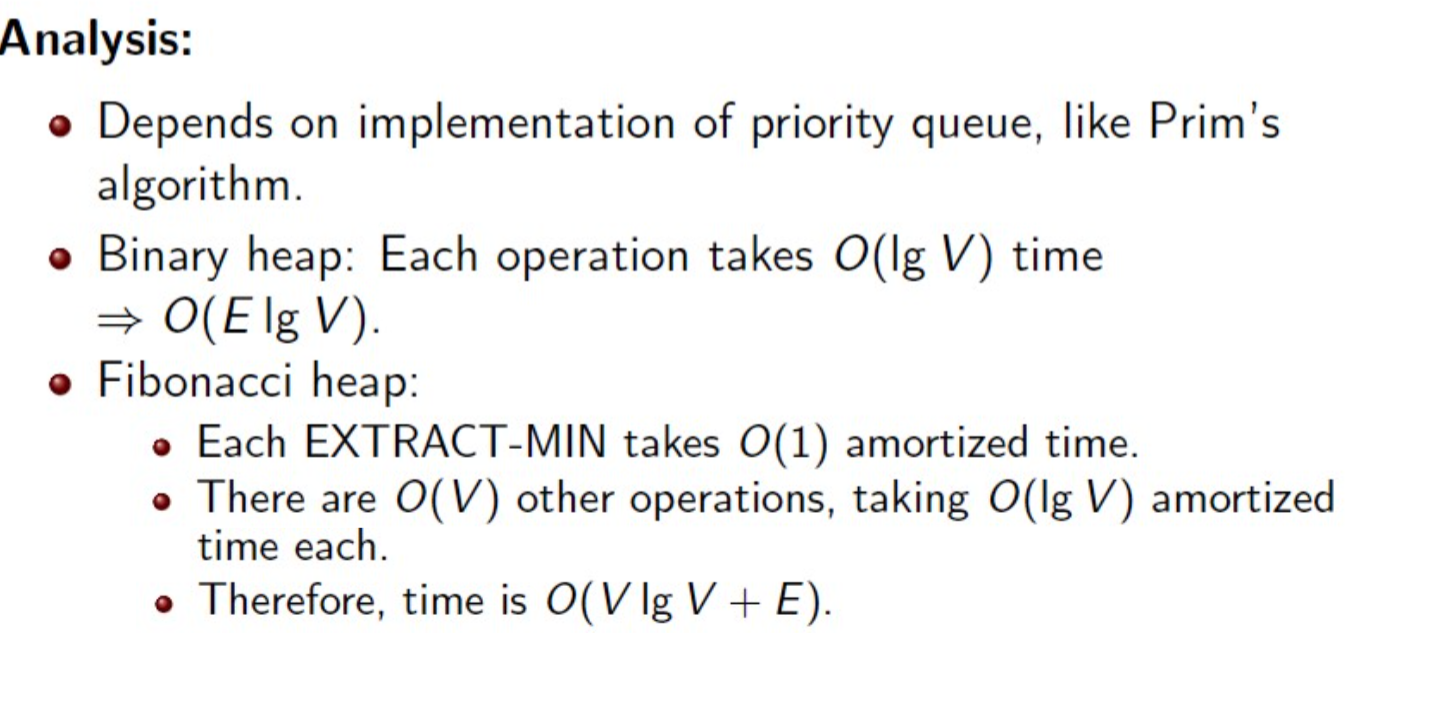
DFS: O(V+E)

Prims: The **time** complexity is O(VlogV + ElogV) = O(ElogV), making it the same as Kruskal's algorithm. However, **Prim's** algorithm can be improved using Fibonacci Heaps (cf Cormen) to O(E + logV).

Krushkal: (ElogV)

Bellmen Ford: O(VE)

Floyd Warshall : O(v^3)

**Dijkastra’s**

**Insertion Sort:**

Average/Worst case = O(n^2)

Best Case = O(n)

**Merger Sort:** Θ(𝑛lg𝑛).

Merge function: Θ(m+n).

**Full binary tree**: each node is either a leaf or has degree of exactly 2. (there is no degree-1 nodes).

**Perfect binary tree**: a full binary tree of height n with exactly 2!−1nodes.

**Complete binary tree**: a perfect binary tree through 𝑛−1with some extra leaf nodes at level 𝑛, all toward the left.

**Heap**: 𝑂(𝑛lg𝑛).

**Quick Sort**:

Worst Case = Θ(𝑛^2)

Best/Average Case = Θ(𝑛lg𝑛)

**Counting Sort:**

Running time Θ(𝑛+𝑘) which is Θ𝑛 if 𝑘=O(n).

Radix Sort: The total running time is Θ(𝑑𝑛+𝑘)⇒Θ(𝑑𝑛), if 𝑘=𝑂(𝑛).

Bucket Sort: Θ(𝑛)+𝑂(𝑛)=Θ(𝑛)

**BST:**

InOrder: O(n)

Worst case: O(n^2)

Best Case: O(nlogn)

We will show that the expected height of a randomly built binary search tree is O(lgn).

**Red Black Tree:**

Balanced: height is O(lg n), where n is the number of nodes. •Operations will take O(lg n) time in the worst case.

Red Black Tree:

Searching: average case: O(lg n) Best case = O(1) worst case = O(n)

Insertion: O(log n)

DFS Edges:

|  |  |  |
| --- | --- | --- |
| **Edge type of uv** | **start times** | **end times** |
| Tree edge | start[u] < start[v] | end[u] > end[v] |
| Back edge | start[u] > start[v] | end[u] < end[v] |
| Forward edge | start[u] < start[v] | end[u] > end[v] |
| Cross edge | start[u] > start[v] | end[u] > end[v] |

https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture3.pdf