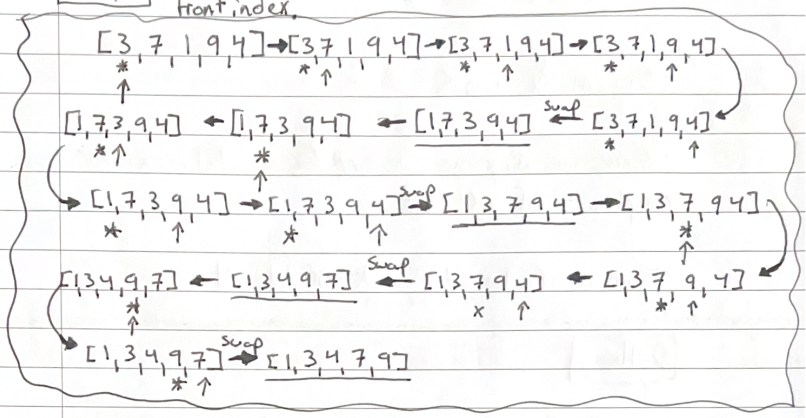
Sorting

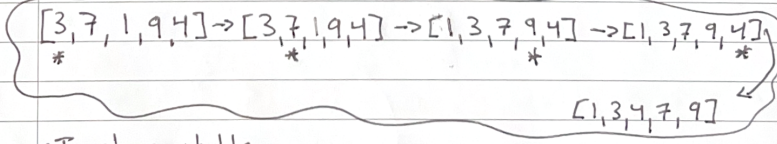
Selection Sort • Select the smallest element & swap it with the current front index.



- In place & not stable

  Example: [4a, 5, 3, 4b, 1] → [1, 3, 4b, 4a, 5]
- $O(n^2)$ no matter what
- Minimal data movement

Insert Sort • Move elements to the left until the element to the left is smaller.



- In place, stable
- Linear if partially sorted
- $O(n^2)$ depends on input

Shell Sort

- Check lst[n] < lst[ht4]
- Insert sort with stride h

h = n/2

[12, 34, 54, 2, 3] → [12, 34, 54, 2, 3] swap [12, 2, 54, 34, 3]

swap [12, 2, 3, 34, 54] ← [12, 2, 3, 34, 54] swap [12, 2, 54, 34, 3]

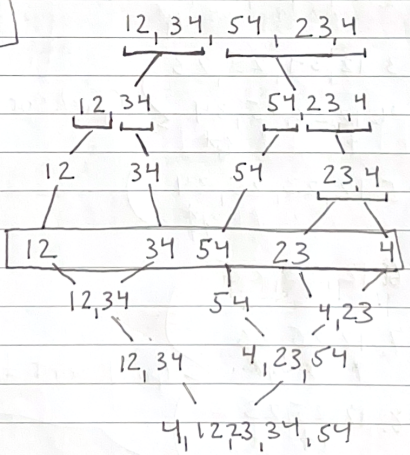
[3, 2, 12, 34, 54] → [3, 2, 12, 34, 54] swap [2, 3, 12, 34, 54]

Gap is now n/4 = h

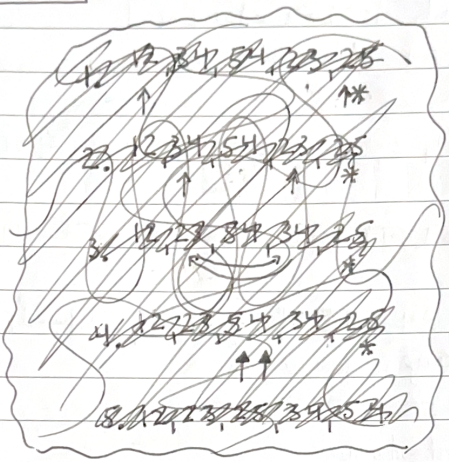
Gap becomes 0 so array is sorted

- In place, not stable
- Run time depends on sequence
- Bad → $O(n^2)$, Good → $O(n^{4/3})$

Merge Sort



Quick sort



[12, 34, 54, 23, 25]

1. 12, 34, 25, 23, 54
2. 12, 34, 25, 23, 54

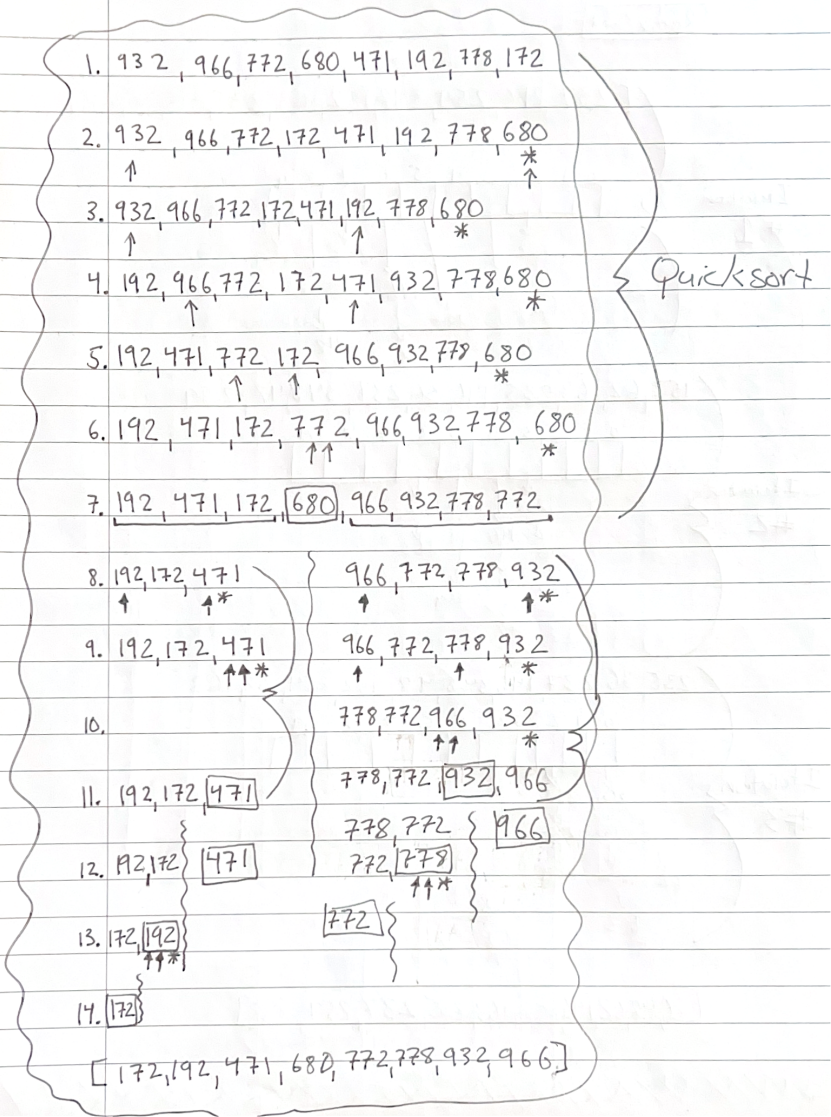
Quicksort #1

3. 12, 34, 25, 23
4. 12, 23, 25, 34
5. 12, 23, 25, 34

Quicksort #2

6. 12, 23, 25
7. 12, 25, 23
8. 12, 25, 23
9. 12, 23, 25

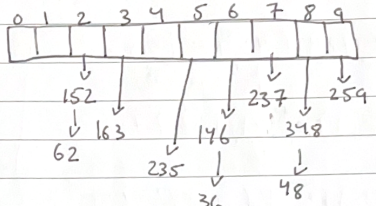
Quicksort #3


1. 932, 966, 772, 680, 471, 192, 778, 172
2. 932, 966, 772, 172, 471, 192, 778, 680
3. 932, 966, 772, 172, 471, 192, 778, 680
4. 192, 966, 772, 172, 471, 932, 778, 680
5. 192, 471, 772, 172, 966, 932, 778, 680
6. 192, 471, 172, 772, 966, 932, 778, 680
7. 192, 471, 172, 680, 966, 932, 778, 772
Quicksort
8. 192, 172, 471
966, 772, 778, 932
9. 192, 172, 471
966, 772, 778, 932
10. 778, 772, 966, 932
11. 192, 172, 471
778, 772, 932, 966
12. 192, 172
471
778, 772
966
772, 778
13. 172, 192
772
14. 172
[172, 192, 471, 680, 772, 778, 932, 966]

Radix Sort

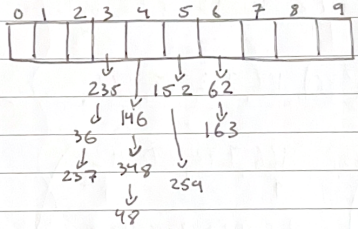
[237, 146, 259, 348, 152, 163, 235, 48, 36, 62]

Iteration #1



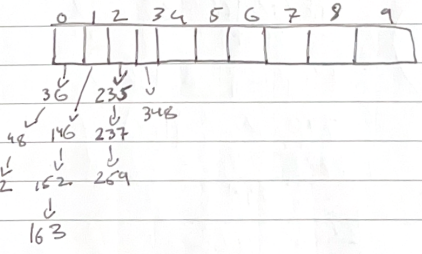
152, 62, 163, 235, 146, 36, 237, 348, 48, 259

Iteration #2



235, 36, 237, 146, 348, 48, 152, 259, 62, 163

Iteration #3



[36, 48, 62, 146, 152, 163, 235, 237, 259, 348]

Graphs

Graph traversal and orderings
• Topological sort ✓
• BFS ✓
• DFS ✓

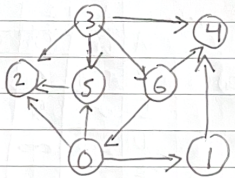
Shortest Path
• Dijkstra ✓
• Bellman-Ford ✓
• Floyd-Warshall ✓

Minimum Spanning tree
• Prim ✓
• Kruskal ✓

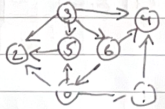
Strongly connected components
• Kosaraju-Sharif

Topological Sort

*Must be DAG



1. DFS for reverse post order



[4, 1, 2, 5, 0, 6, 3]

2. Reverse it → 3 6 0 5 2 1 4 ✓

Dijkstra

* Greedy
* No negative edgeweights
* Single source shortest path $O(N^2)$

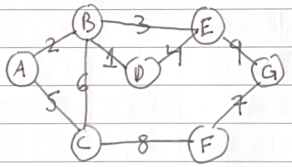
1. Set each point's initial cost value to ∞ besides for the start point.

2. Search from the start for unseen vertexes.

3. If the cost of curr vertex + cost(v,w) is less then the current cost to get to w update w.

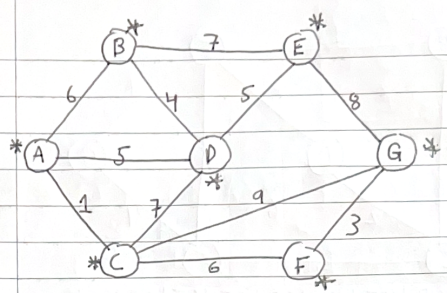
4. From the candidate points choose the one with the lowest cost. Now we know that the path taken is the shortest path.

5....



looking from	A	A	B	D	...
A	0	0	0	0	...
B	∞	2	2	2	...
C	∞	5	5	5	...
D	∞	∞	3	3	...
E	∞	∞	5	5	...
F	∞	∞	∞	∞	...
G	∞	∞	∞	∞	...

...looking from	C	E	F	G
A	0	0	0	0
B	2	2	2	2
C	5	5	5	5
D	3	3	3	3
E	5	5	5	5
F	13	13	13	13
G	∞	14	14	14



Curr	A	A	C	D	B	F	E	G
A	0	0	0	0	0	0	0	0
B	∞	6	6	6	6	6	6	6
C	∞	1	1	1	1	1	1	1
D	∞	5	5	5	5	5	5	5
E	∞	∞	∞	10	10	10	10	10
F	∞	∞	7	7	7	7	7	7
G	∞	∞	10	10	10	10	10	10

Bellman Ford * At most N-1 iterations of Relaxing edges

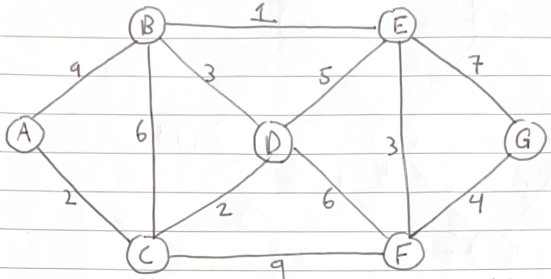
1. Init all costs to ∞ except for curr = 0. * Dynamic

2. Calculate the cost to get to a neighbor from curr. If the cost is less update it.

3. Check the cost in the reverse direction. e.g A→B, B→A and maybe update it.

* Can detect negative cycles

* Works with negative edge weights



Iteration #1

	AA	AB	AC	CB	BE	BD	CD	CF	DF	DE	EG	FG
A	0	0	0	0	0	0	0	0	0	0	0	0
B	∞	9	9	8	8	8	8	8	8	8	8	8
C	∞	∞	2	2	2	2	2	2	2	2	2	2
D	∞	∞	∞	∞	∞	11	4	4	4	4	4	4
E	∞	∞	∞	∞	9	9	9	9	9	9	9	9
F	∞	∞	∞	∞	∞	∞	∞	11	10	10	10	10
G	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	16	14

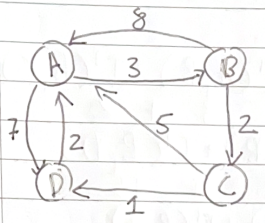
Iteration #2

	AB	AC	CB	BE	BD	CD	CF	DF	DE	EG	FG
A	0	0	0	0	0	0	0	0	0	0	0
B	8	8	8	8	8	7	7	7	7	7	7
C	2	2	2	2	2	2	2	2	2	2	2
D	4	4	4	4	4	4	4	4	4	4	4
E	9	9	9	9	9	9	9	9	9	9	9
F	10	10	10	10	10	10	10	10	10	10	10
G	14	14	14	14	14	14	14	14	14	14	14

* Keep Going Until No Updates

...

Floyd-Warshall * Dynamic Prog
* All pairs shortest Path



I^0 =

	A	B	C	D
A	0	3	∞	7
B	8	0	2	∞
C	5	∞	0	1
D	2	∞	∞	0

Curr = None

I' =

	A	B	C	D
A	0	3	∞	7
B	8	0	2	15
C	5	8	0	1
D	2	5	∞	0

Curr = A

B→C: $I^0(B,C)$ or $I^0(B,A)+I^0(A,C)$
2 < 8 + ∞

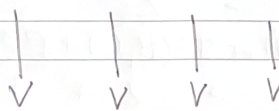
B→D: $I^0(B,D)$ or $I^0(B,A)+I^0(A,D)$
∞ = 8 + 7

C→B: $I^0(C,B)$ or $I^0(C,A)+I^0(A,B)$
∞ > 5 + 3

C→D: $I^0(C,D)$ or $I^0(C,A)+I^0(A,D)$
1 < 5 + 7

D→B: $I^0(D,B)$ or $I^0(D,A)+I^0(A,B)$
∞ > 2 + 3

D→C: $I^0(D,C)$ or $I^0(D,A)+I^0(A,C)$
∞ = 2 + ∞



$I^2 =$

	A	B	C	D
A	0	3	5	7
B	8	0	2	15
C	5	8	0	1
D	2	5	7	0

curr = B

A -> C: $I^1(A,C)$ or $I^1(A,B) + I^1(B,C)$
∞ > 3 + 2

A -> D: $I^1(A,D)$ or $I^1(A,B) + I^1(B,D)$
7 < 3 + 15

C -> A: $I^1(C,A)$ or $I^1(C,B) + I^1(B,A)$
5 < 8 + 8

C -> D: $I^1(C,D)$ or $I^1(C,B) + I^1(B,D)$
1 < 8 + 15

D -> A: $I^1(D,A)$ or $I^1(D,B) + I^1(B,A)$
2 < 5 + 8

D -> C: $I^1(D,C)$ or $I^1(D,B) + I^1(B,C)$
∞ > 5 + 2

$I^3 =$

	A	B	C	D
A	0	3	5	6
B	7	0	2	3
C	5	8	0	1
D	2	5	7	0

curr = C

A -> B: $I^2(A,B)$ or $I^2(A,C) + I^2(C,B)$
3 < 5 + 8

A -> D: $I^2(A,D)$ or $I^2(A,C) + I^2(C,D)$
7 > 5 + 1

B -> A: $I^2(B,A)$ or $I^2(B,C) + I^2(C,A)$
8 > 2 5

B -> D: $I^2(B,D)$ or $I^2(B,C) + I^2(C,D)$
15 > 2 + 1

$D \to A: I^2(D,A)$ or $I^2(D,C) + I^2(C,A)$

$2 \quad < \quad 7 + 5$

$D \to B: I^2(D,B)$ or $I^2(D,C) + I^2(C,B)$

$5 \quad < \quad 7 + 8$

$I^4 =$

	A	B	C	D
A	0	3	5	6
B	5	0	2	3
C	3	6	0	1
D	2	5	7	0

carr = D

$A \to B: I^3(A,B)$ or $I^3(A,D) + I^3(D,B)$

$3 \quad < \quad 6 + 5$

$A \to C: I^3(A,C)$ or $I^3(A,D) + I^3(D,C)$

$5 \quad < \quad 6 + 7$

$B \to A: I^3(B,A)$ or $I^3(B,D) + I^3(D,A)$

$7 \quad > \quad 3 + 2$

$B \to C: I^3(B,C)$ or $I^3(B,D) + I^3(D,C)$

$2 \quad < \quad 3 + 7$

$C \to A: I^3(C,A)$ or $I^3(C,D) + I^3(D,A)$

$5 \quad > \quad 1 + 2$

$C \to B: I^3(C,B)$ or $I^3(C,D) + I^3(D,B)$

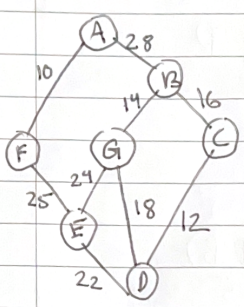
$8 \quad > \quad 1 + 5$

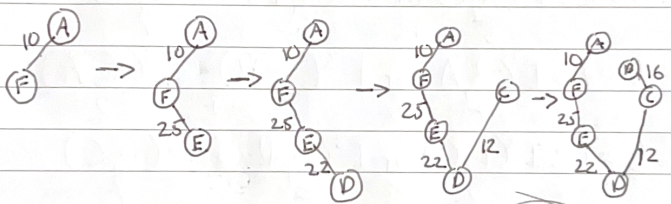
* No negative cycles

Prim

* Greedy
* Must not have disjoint sets
* Min cost spanning tree

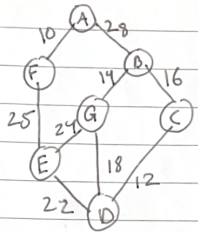
1. Grab smallest cost edge
2. Add smallest cost edge that forms a tree





Kruskal

* Greedy
* $O(M \cdot K)$



1. Select smallest cost edge
2. Keep doing that for edges that do not form a cycle.