CSE 373 Section Handout #9
Sorting Algorithm Reference

stooge sort:

Swap first/last if out of order, then stooge-sort the first 2/3, then last 2/3, then first 2/3 again.

A total of 40 recursive calls are made! Ouch.

index	0	1	2	3	4	5
value	9	6	2	4	1	5
call #1	**5**	6	2	4	1	**9**
#2	**4**	6	2	**5**		
#3	**2**	6	**4**			
#4	2	6				
#5		**4**	**6**			
#6	2	4				
#7		4	6	5		
#8		4	6			
#9			**5**	**6**		
#10		4	5			
#11-14	**2**	4	5			
#15			5	6	1	9

... calls 12-14 omitted (no swaps made)

- $O(N^{2.7095...})$
- Silly; slower than bubble sort.

heap sort:

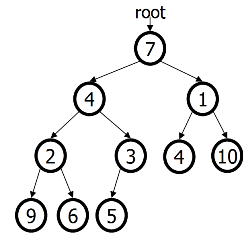
Turn array into a max-heap, then remove-max in place moving each root to the end until the array is sorted.

index	0	1	2	3	4	5	6	7	8	9
value	7	4	1	2	3	4	10	9	6	5

index	0	1	2	3	4	5	6	7	8	9
value	7	4	1	2	**5**	4	10	9	6	**3**

index	0	1	2	3	4	5	6	7	8	9
value	7	4	1	**9**	5	4	10	**2**	6	3

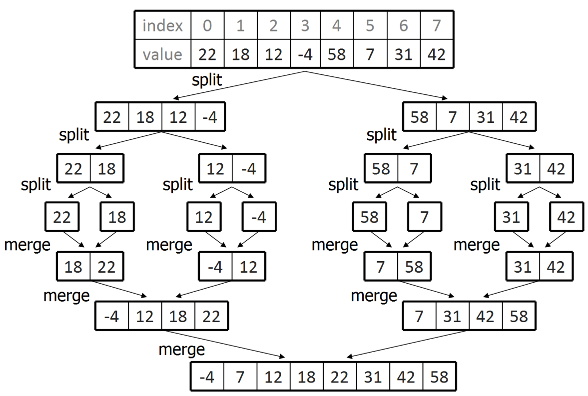
...



- $O(N \log N)$ in all cases.
- Slower than merge sort, faster than shell sort.

merge sort:

Split array in half, sort the halves, then merge the sorted halves back together.



- $O(N \log N)$ in all cases; $O(N)$ memory used.
- A fast, general purpose, "stable" sort.

quick sort:

Choose some element as the "pivot". Partition the array into two groups: elements < pivot, and ≥ pivot. Then recursively repeat the process on each group.

index	0	1	2	3	4	5	6	7	8	9	
value	**65**	23	81	43	92	39	57	16	75	32	choose pivot=65
	32	23	81	43	92	39	57	16	75	**65**	swap pivot (65) to end
	32	23	**16**	43	92	39	57	**81**	75	65	swap 81, 16
	32	23	16	43	**57**	39	**92**	81	75	65	swap 57, 92
	32	23	16	43	57	39	92	81	75	65	
	32	23	16	43	57	39	**65**	81	75	**92**	swap pivot back in

recursively quicksort each half

32	23	16	43	57	39	pivot=32
39	23	16	43	57	**32**	swap to end
16	23	**39**	43	57	32	swap 39, 16
16	23	**32**	43	57	**39**	swap 32 back in

...

81	75	92	pivot=81
92	75	**81**	swap to end
75	92	81	swap 92, 75
75	**81**	**92**	swap 81 back in

...

- $O(N \log N)$ average, $O(N^2)$ worst-case.
- Choosing pivot poorly can hurt performance.

bucket sort: *(integers only)*

Create array of tallies. Tally occurrences of int value *i* in index [*i*]. Use tallies to regenerate sorted elements.

- input array *a*:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	7	4	1	2	5	4	10	9	1	4	7	8	9	8	9	4	4

- create array of tallies:

index	0	1	2	3	4	5	6	7	8	9	10
value	0	2	1	0	5	1	0	2	2	3	1

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	1	1	2	4	4	4	4	4	5	7	7	8	8	9	9	9	10

- use tallies to generate sorted contents of *a*

- $O(M + N)$ for N ints in range [0 .. M); ~$O(N)$
- Very fast! But works only on fixed-range ints.

radix sort: *(integers/strings)*

Perform a pass of bucket sort for each digit / character, from least to most significant digit.

- input array *a*:

index	0	1	2	3	4	5	6	7	8	9	10	11
value	714	128	206	34	722	8	142	533	646	29	240	373

- create array of queues, ordered by last digit:

index	0	1	2	3	4	5	6	7	8	9
value	240		722, 142	533, 373	714, 34		206, 646		8, 128	29

index	0	1	2	3	4	5	6	7	8	9	10	11
value	240	722	142	533	373	714	034	206	646	008	128	029

...

index	0	1	2	3	4	5	6	7	8	9	10	11
value	206	008	714	722	128	029	533	034	240	142	646	373

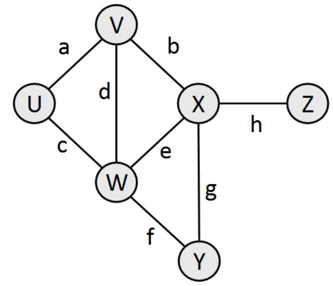
...

index	0	1	2	3	4	5	6	7	8	9	10	11
value	008	029	034	128	142	206	240	373	533	646	714	722

- $O(N)$ assuming number of digits is small
- Very fast! Works with ints and strings.

CSE 373 Section Handout #9
Graph Reference

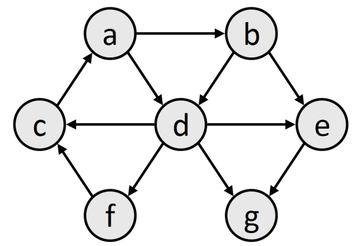
graph: A data structure containing:
a set of **vertices** V, (sometimes called nodes)
a set of **edges** E, where an edge#represents a connection between 2 vertices.



degree: number of edges touching a given vertex.

path: A path from vertex a to b is a sequence of edges that can be followed starting from a to reach b.
can be represented as vertices visited, or edges taken

path length: Number of vertices or edges contained in the path.

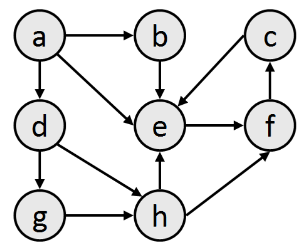


neighbor or **adjacent**: Two vertices connected directly by an edge.

reachable: Vertex a is reachable from b if a path exists from a to b.

connected: A graph is connected if every vertex is reachable from any other.
strongly connected: When every vertex has an edge to every other vertex.

cycle: A path that begins and ends at the same node.
acyclic graph: One that does not contain any cycles.
loop: An edge directly from a node to itself.

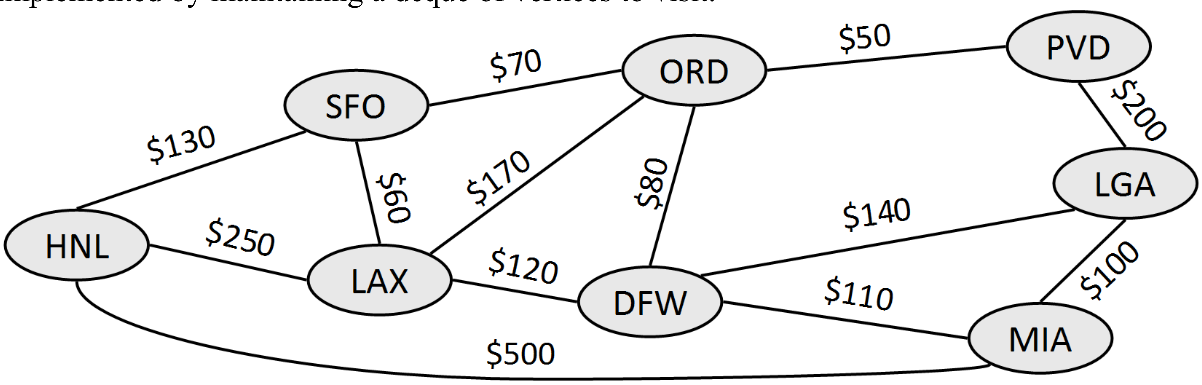


weight: Cost associated with a given edge.
weighted graph: One where edges have weights *(see graph below)*.

directed graph ("digraph"): One where edges are one-way connections.

depth-first search (DFS): Finds a path between two vertices by exploring each possible path as far as possible before backtracking.
Often implemented recursively.

breadth-first search (BFS): Finds a path between two nodes by taking one step down all paths and then immediately backtracking.
Often implemented by maintaining a deque of vertices to visit.



CSE 373 Section Handout #9

The problems on this page refer to the following arrays:

```
index     0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
a)          {59, 15,  6, 28, 32, -7, 41,  8}
b)          {19, 27,  6, 34, 46,  8, −4, 13, 51, 11}
c)        {315, 88, 21, 149, 308, 6, 708, 411, 79, 116, 265, 400}
```

1. **Merge Sort Tracing**
   Trace the execution of the *merge sort* algorithm over array a) above. Show each pass of the algorithm and the splitting/merging of the array, until the array is sorted.

2. **Quick Sort Tracing**
   Trace the execution of the *quick sort* algorithm over array b) above. Use the *first element* as the pivot. Show each pass of the algorithm, with the pivot selection and partitioning, and the state of the array as/after the partition is performed, until the array is sorted. You do not need to show partitioning calls over a single element, because there is nothing to do.

3. **Radix Sort Tracing**
   Trace the execution of the *radix sort* algorithm over array c) above. Show each pass of the algorithm and its array of tallies, then show the state of the array after the pass has been performed, until the array is sorted.

For more practice later, try performing the algorithms over the other arrays and see the results.

CSE 373 Section Handout #9

4. graph properties

For the graphs shown below, answer the following questions: *(sorry for the crappy graph drawings!)*

a) Which graphs are directed, and which are undirected?

b) Which graphs are weighted, and which are unweighted?

c) Which graphs are connected, and which are not? Is any graph strongly connected?

d) Which graphs are cyclic, and which are acyclic?

e) What is the degree of each vertex? (If it is directed, what is the in-degree and out-degree?)

```
Graph 1:

A --► B ◄-- C
|     |     ▲
|     |     |
▼     ▼     |
D ◄-- E --► F
|     ▲     ▲
|     |     |
▼     |     |
G ◄-► H ◄-- I
```

```
Graph 2:

A     B----C
|     |\_
|     |  \_
D-----E    F
```

```
Graph 3:

A ◄--► B ◄-- C
|      ▲
▼      |
D ◄--► E
```

```
Graph 4:

   3
 A---B
 |  /
5| /2
 |/
 C   D---E
       8
```

```
Graph 5:

A-----B
|\   /|
| \ / |
|  +  |
| / \ |
|/   \|
C-----D
```

```
Graph 6:
     ______________
    /     8        \
   /  4        1    ▼    7
  A ◄---- B ◄---► C ----► D
  ▲       ▲       |       ▲
 1|      2|       |5      |
  |       |       |     1/
  ▼   2   ▼   3   ▼      /
  E ◄---► F ◄---- G ----/
```

5. depth-first search (DFS)

Write the paths that a depth-first search would find from vertex A to all other vertices:

- in Graph 1
- in Graph 6

If a given vertex is not reachable from vertex A, write "no path" or "unreachable".

6. breadth-first search (BFS)

Write the paths that a breadth-first search would find from vertex A to all other vertices:

- in Graph 1
- in Graph 6

Which paths are shorter than the ones found by DFS in the previous problem?

7. minimum weight paths

Which paths found by DFS and BFS on Graph 6 in the previous problems are not minimal weight?
What are the minimal weight paths from vertex A to all other nodes? *(Just inspect the graph manually.)*

CSE 373 Section Handout #9
Solutions

1. merge sort

```
index      0   1   2   3    4   5   6   7
original {59, 15,  6, 28,  32, -7, 41,  8}
split    {59, 15,  6, 28},{32, -7, 41,  8}
split    {59, 15},{6, 28}
split    {59}{15}
merge    {15, 59}
split             {6, 28}
split             {6}{28}
merge             {6, 28}
merge    { 6, 15, 28, 59}
split                     {32, -7},{41, 8}
split                     {32}{-7}
merge                     {-7, 32}
split                              {41}{8}
merge                              {8, 41}
merge                     {-7,  8, 32, 41}
merge    {-7,  6,  8, 15,  28, 32, 41, 49}
```

2. quick sort

```
index      0   1   2   3   4   5   6   7   8   9
original {19, 27,  6, 34, 46,  8, -4, 13, 51, 11}
         {11, 27,  6, 34, 46,  8, -4, 13, 51, 19}  pivot (19) to end
              13                      27           partitioning
                      -4          34
                           8  46
          11, 13,  6, -4,  8, 19, 34, 27, 51, 46   swap pivot back in

           8, 13,  6, -4, 11                       pivot (11) to end
              -4      13                           partitioning
           8, -4,  6, 11, 13                       swap pivot back in

           6, -4,  8                               pivot (8) to end
          -4   6                                   partitioning
          -4,  6,  8
                                  46, 27, 51, 34   pivot (34) to end
                                  27  46           partitioning
                                  27, 34, 51, 46   swap pivot back in
                                          46, 51   pivot (51) to end
                                                   nothing to do
         {-4,  6,  8, 11, 13, 19, 27, 34, 46, 51}
```

3. radix sort

```
 index        0    1    2    3    4    5    6    7    8    9   10   11
 original   {315,  88,  21, 149, 308,   6, 708, 411,  79, 116, 265, 400}

 1s bucket   400   21,                 315,   6,        88, 149,
                  411                  265  116        308,  79
                                                       708
            {400,  21, 411, 315, 265,   6, 116,  88, 308, 708, 149,  79}

10s bucket   400, 411,  21       149       265   79   88
              _6, 315,
             308, 116
             708
            {400,   6, 308, 708, 411, 315, 116,  21, 149, 265,  79,  88}

100s bucket  _ 6, 116, 265  308, 400,           708
             _21, 149       315  411
             _79,
              88
             {6,  21,  79,  88, 116, 149, 265, 308, 315, 400, 411, 708}
```

CSE 373 Section Handout #9
Solutions, continued

4.

```
...
Graph 1: directed, unweighted, not connected, cyclic
    degrees: A=(in 0 out 2), B=(in 2 out 1), C=(in 1 out 1), D=(in 2 out 1),
             E=(in 2 out 2), F=(in 2 out 1), G=(in 2 out 1), H=(in 2 out 1),
             I=(in 0 out 2)

Graph 2: undirected, unweighted, connected, acyclic
    degrees: A=1, B=3, C=1, D=2, E=2, F=1

Graph 3: directed, unweighted, not connected, cyclic
    degrees: A=(in 1 out 2), B=(in 3 out 1), C=(in 0 out 1),
             D=(in 2 out 1), E=(in 1 out 2)

Graph 4: undirected, weighted, not connected, cyclic
    degrees: A=2, B=2, C=2, D=1, E=1

Graph 5: undirected, unweighted, strongly connected, cyclic
    degrees: A=3, B=3, C=3, D=3

Graph 6: directed, weighted, connected, cyclic
    degrees: A=(in 2 out 2), B=(in 2 out 3), C=(in 2 out 3), D=(in 2 out 0),
             E=(in 2 out 2), F=(in 3 out 2), G=(in 1 out 2)
```

5. DFS

Graph 1
A to B: {A, B}
A to C: {A, B, E, F, C}
A to D: {A, B, E, D}
A to E: {A, B, E}
A to F: {A, B, E, F}
A to G: {A, B, E, D, G}
A to H: {A, B, E, D, G, H}
A to I: no path

Graph 6
A to B: {A, C, B}
A to C: {A, C}
A to D: {A, C, D}
A to E: {A, C, B, F, E}
A to F: {A, C, B, F}
A to G: {A, C, G}

6. BFS *(shorter paths in bold)*

Graph 1
A to B: {A, B}
A to C: {A, B, E, F, C}
A to D: **{A, D}**
A to E: {A, B, E}
A to F: {A, B, E, F}
A to G: **{A, D, G}**
A to H: **{A, D, G, H}**
A to I: no path

Graph 6
A to B: {A, C, B}
A to C: {A, C}
A to D: {A, C, D}
A to E: **{A, E}**
A to F: **{A, E, F}**
A to G: {A, C, G}

7. minimum weight paths

Graph 6
A to B: **{A, E, F, B}**, weight=5
A to C: **{A, E, F, B, C}**, weight=6
A to D: **{A, E, F, B, C, G, D}**, weight=12
A to E: {A, E}, weight=1
A to F: {A, E, F}, weight=3
A to G: **{A, E, F, B, C, G}**, weight=11