Sorting

Selection Sort

- For each pass, select the next largest/smallest number and put it in its place
  - Ex: Look for the smallest item, swap it with the item in the first position
- N-1 passes

Selection Sort

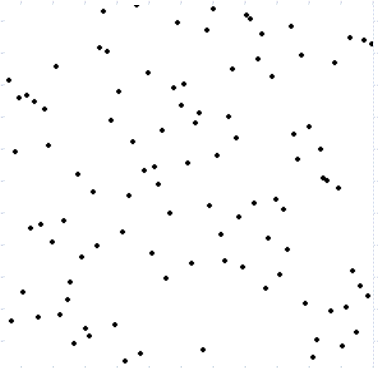
pass	12	4	2	26	7	30	19	11	21

Selection Sort

Running time = $O(n^2)$

pass	12	4	2	26	7	30	19	11	21
P=1	**2**	4	**12**	26	7	30	19	11	21
P=2	2	**4**	12	26	7	30	19	11	21
P=3	2	4	**7**	26	**12**	30	19	11	21
P=4	2	4	7	**11**	12	30	19	**26**	21
P=5	2	4	7	11	**12**	30	19	26	21
P=6	2	4	7	11	12	**19**	**30**	26	21
P=7	2	4	7	11	12	19	**21**	26	**30**
P=8	2	4	7	11	12	19	21	**26**	30

Selection Sort



Bubble Sort

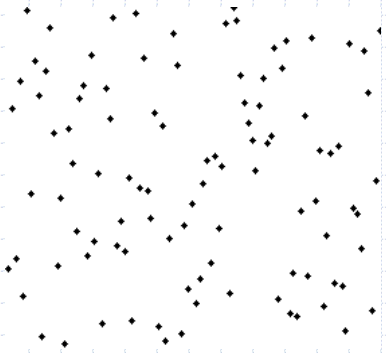
- Compare adjacent items and exchange them if they are out of order
- N-1 passes
- Naïve implementation
  - Continue for N-1 passes
- Smart implementation
  - If no swapping on previous pass, done
  - On each pass, traverse 1 less item
    - Unsorted portion becomes 1 less

Bubble Sort

pass	10	14	2	16	7	21	8
P=1	10	2	14	7	16	8	21
P=2	2	10	7	14	8	16	21
P=3	2	7	10	8	14	16	21
P=4	2	7	8	10	14	16	21
P=5	2	7	8	10	14	16	21

Running time = $O(n^2)$ for all implementations
however smart implementation will run faster

Bubble Sort



Insertion Sort

- Make N-1 passes
- For P=1 through N-1, ensure that the items in positions 0 through P are sorted
- Insert next item into correct position in already sorted set, now set is one bigger

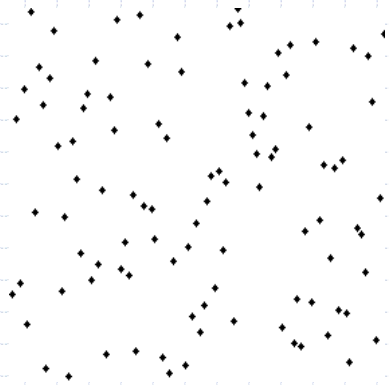
Insertion Sort

pass	34	26	8	61	17	51	32	9	22

Insertion Sort

pass	34	26	8	61	17	51	32	9	22
P=1	26	34	8	61	17	51	32	9	22
P=2	8	26	34	61	17	51	32	9	22
P=3	8	26	34	61	17	51	32	9	22
P=4	8	17	26	34	61	51	32	9	22
P=5	8	17	26	34	51	61	32	9	22
P=6	8	17	26	32	34	51	61	9	22
P=7	8	9	17	26	32	34	51	61	22
P=8	8	9	17	22	26	32	34	51	61

Insertion Sort



Insertion Sort

Running time = $O(n^2)$ $O(n)$ if presorted
Worst case - reverse order

Merge Sort

- Divide and conquer
- Merge sets of sorted lists
- Recursive sorting algorithm
  - Divide array in half then merge sort each half

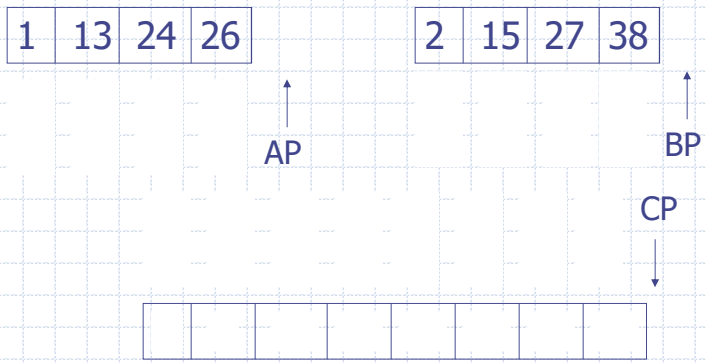
Divide & Conquer Algorithms

- Divide
  - Break down a problem into 2 or more sub-problems of the same type
  - Recursively solve each sub-problem until a sub-problem can be solved directly (base case)
- Conquer
  - Combine solutions to sub-problems to solve original problem (merge)

Merge Sort - Merge Step

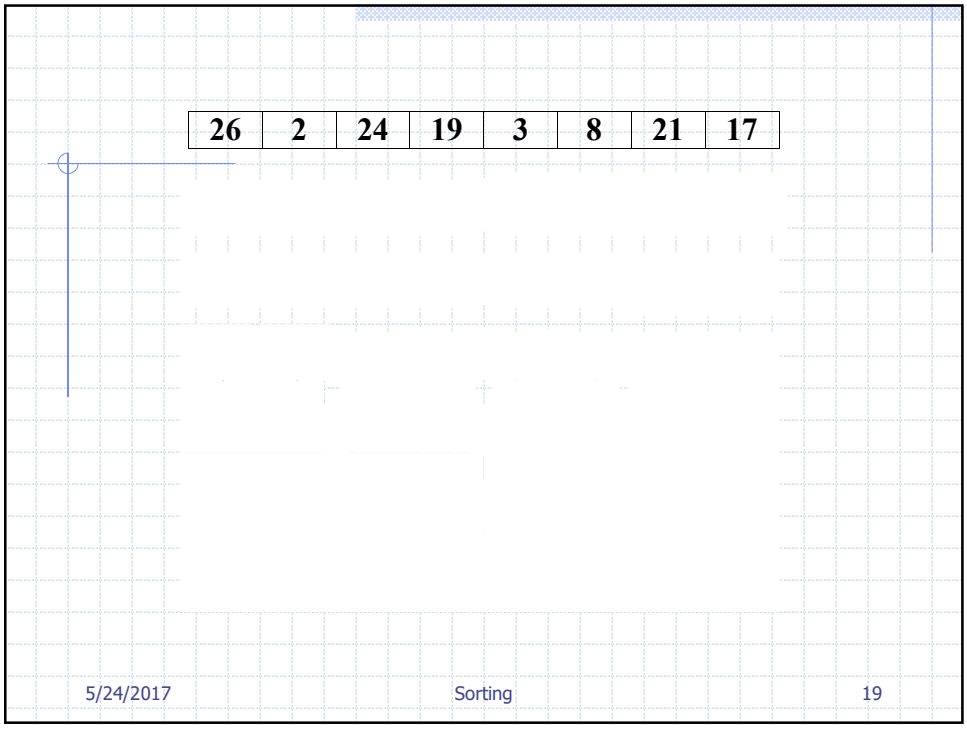
- Takes 2 sorted arrays and merges them to 1 sorted array
  - Input = 2 arrays A and B
  - Output = array C
  - Three pointers - AP, BP, and CP
    - Initially set to the beginning of each array
  - C[CP] = min (A[AP], B[BP])
  - Running time O(n)

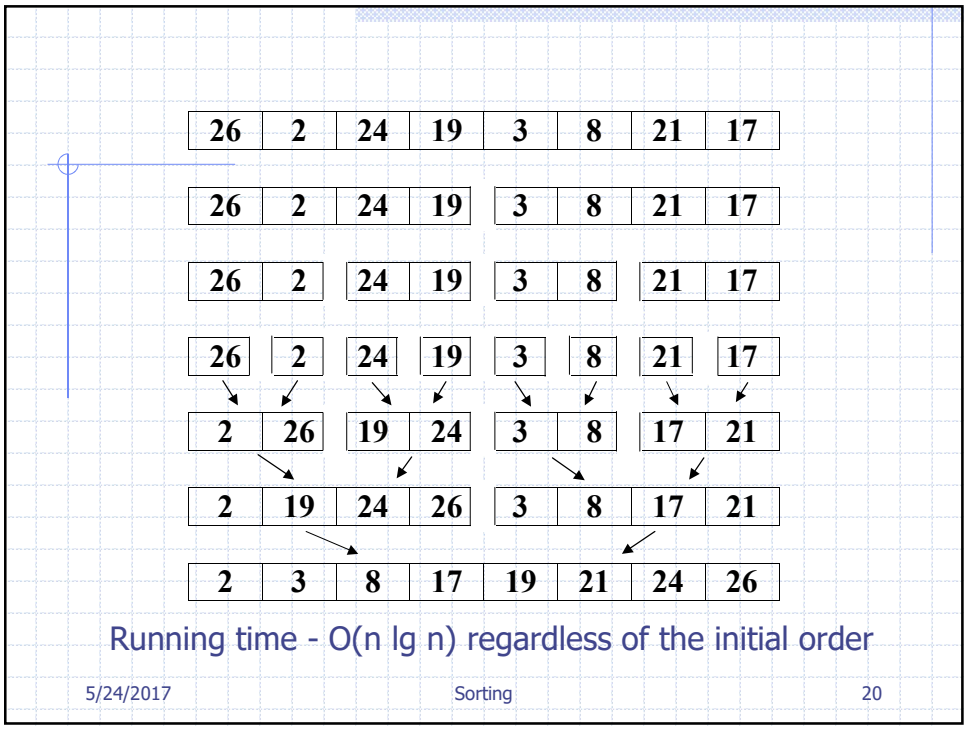
Merge Sort - Merge Step



Merge Sort

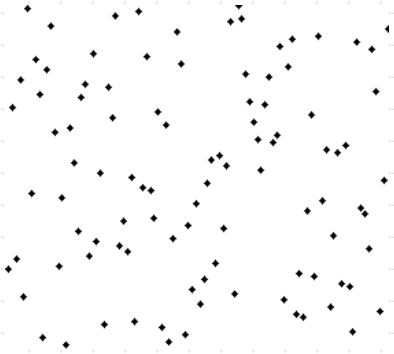
- Recursively sort left half
- Recursively sort right half
- Merge two sorted arrays





Running time - O(n lg n) regardless of the initial order

Merge Sort

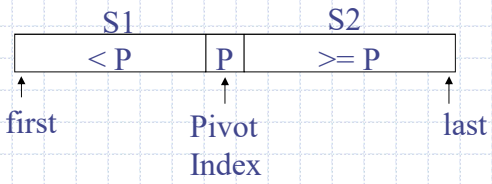


Online resources

- http://www.sorting-algorithms.com/
- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- http://sorting.at/
- Sound of sorting
  - https://www.youtube.com/watch?v=kPRA0W1kECg

Quick Sort

- Partitions an array so items left of the pivot are less than the pivot and items right of the pivot are greater than the pivot
- Pivot is in the correct location

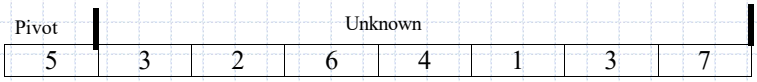


Quick Sort

```
Qsort (A, first, last)
        pivot = partition ( A, first, last )
        if ( first != pivot )
                Qsort ( A, first, pivot-1 )
        if ( last != pivot )
                Qsort ( A, pivot+1, last )
```

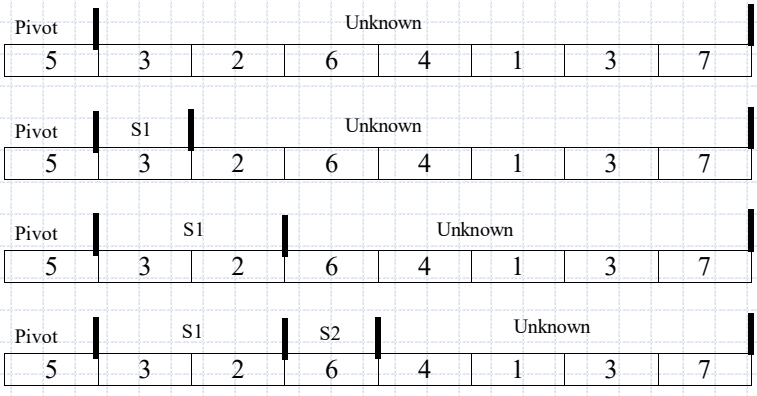
Quick Sort - Partition Step

First pass:

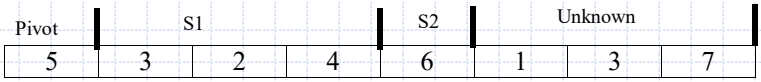


Quick Sort - Partition Step

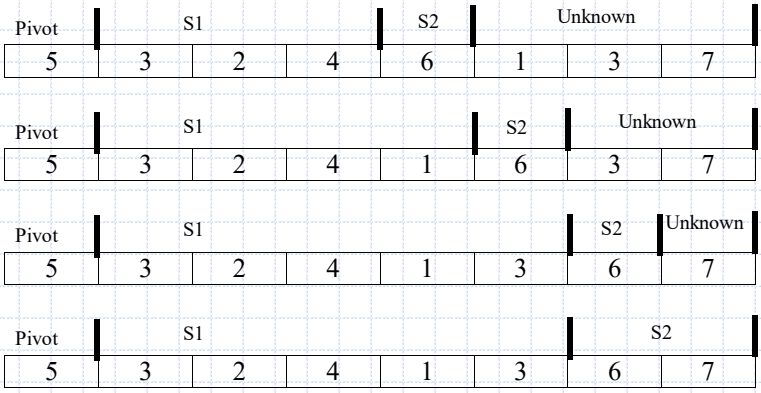
First pass:



Quick Sort - Partition Step



Quick Sort - Partition Step



Quick Sort - Partition Step

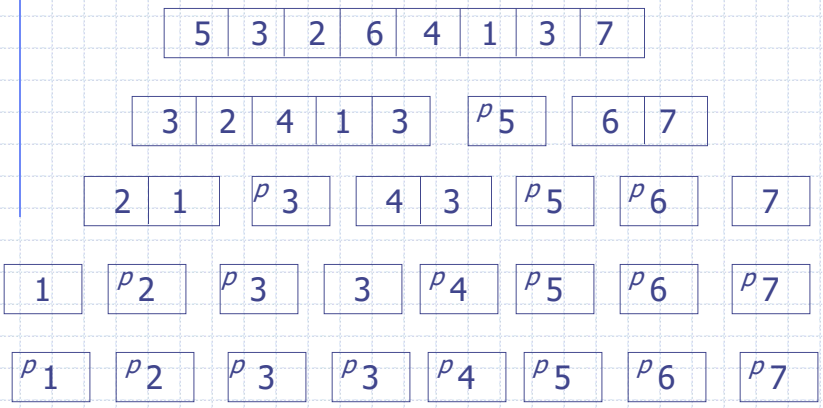
Now perform Qsort of S1 and S2

Running Time - Worst case $O(n^2)$
Average case O(n lg n)

Quick Sort

5	3	2	6	4	1	3	7

Quick Sort



Shell Sort

- Named after Donald Shell
- Compare distant items then decrease distance
- Diminishing increment sort
- Multiple passes, each time sorts a set of numbers

Shell Sort

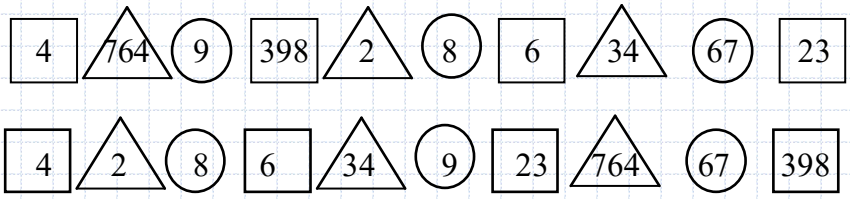
- Generalization of insertion sort
  - An inefficiency of insertion sort is moving elements only 1 position at a time
    - Shell sort allows elements to take bigger jumps toward eventual position
  - Insertion sort most efficient if almost sorted - O(n)
    - Last step is just insertion sort, but elements are almost sorted

Shell Sort

- With each pass, set size gets larger and the number of sets gets smaller
  - Last set contains entire list
- Items contained in set are not necessarily contiguous
  - If there are i sets then each set is composed of every i-th element

Shell Sort

- For i = 3



Shell Sort

- Increment sequence
  - h_k, h_{k-1}, h_{k-2} , … h_1 for h_1=1
  - Each is called a phase
- After each phase, all elements spaced h_k apart are sorted
- $h_k = N/2$ $h_{k-1} = h_k/2$ for each pass k

Shell Sort

	0	1	2	3	4	5	6	7	8
Pass	81	94	11	96	12	35	17	95	28

Shell Sort

	0	1	2	3	4	5	6	7	8
Pass	81	94	11	96	12	35	17	95	28
4 - sort	12	35	11	95	28	94	17	96	81
2 – sort	11	35	12	94	17	95	28	96	81
1 – sort	11	12	17	28	35	81	94	95	96

Running time depends on increment sequence - $O(n^2)$
Runs faster than insertion sort.

Radix Sort

- Forming groups and then combining them to sort a collection of data
- Sort based on value at location one, the value at location two, and so on
- Example
  - Sorting words alphabetically
  - Sorting numbers - treat numbers like string padded with zeros

Radix Sort

64, 8, 217, 512, 27, 728, 40, 1, 343, 125, 313

Pass	0	1	2	3	4	5	6	7	8	9
P=1										
P=2										
P=3										

Radix Sort

64, 8, 217, 512, 27, 728, 40, 1, 343, 125, 313

Pass	0	1	2	3	4	5	6	7	8	9
P=1	40	1	512	343 313	64	125		217 27	8 728	
P=2	1 8	512 313 217	125 27 728		40 343		64			
P=3	1 8 27 40 64	125	217	313 343		512		728		

Running time O(m*N) where m is the longest length and could be very large

Sorting Stability

- Duplicate numbers appear in the same order in the output as they did in the input
  - Think about the sorting algorithms - which ones are stable?

Which sorting algorithm to use?

- How many keys will you be sorting?
  - For small amounts of data n <= 100, quadratic algorithms are fine
  - For larger amounts of data use O(n lg n)
- Will there be duplicate keys in the data?
  - How are ties broken? Is there a secondary key?
  - Should the sort be stable?

Which sorting algorithm to use?

- What do you know about your data?
  - Is the data partially sorted?
    - Insertion sort works better for sorted data
  - Do you know the distribution of the keys?
    - Radix sort might be bad if a lot of clumping
  - Are your keys very long or hard to compare?
    - Radix sort to avoid expensive comparisons
  - Is the range of possible keys very small?

Which sorting algorithm to use?

- Do you have to worry about disk accesses?
  - Does the entire set fit in memory?
  - Read data into a tree and then inorder traversal might be the best.
- How much time do you have to write and debug your routine?
  - Insertion sort, heap sort, or quicksort