CSE408
Bubble sort
Maximum & Minimum

Lecture #16

Why Study Sorting Algorithms?

- There are a variety of situations that we can encounter
  - Do we have randomly ordered keys?
  - Are all keys distinct?
  - How large is the set of keys to be ordered?
  - Need guaranteed performance?

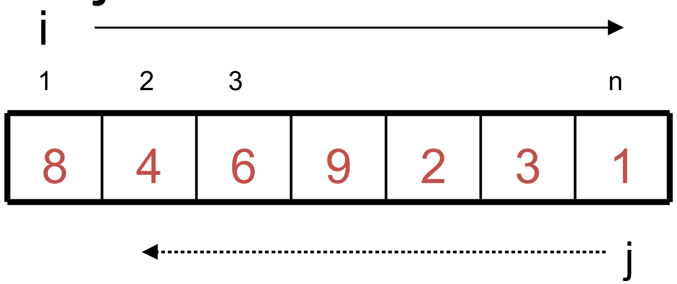
- Various algorithms are better suited to some of these situations

Some Definitions

- Internal Sort
  - The data to be sorted is all stored in the computer's main memory.
- External Sort
  - Some of the data to be sorted might be stored in some external, slower, device.
- In Place Sort
  - The amount of extra space required to sort the data is constant with the input size.

Bubble Sort

- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

Example

8	4	6	9	2	3	1

i = 1 j

8	4	6	9	2	1	3

i = 1 j

8	4	6	9	1	2	3

i = 1 j

8	4	6	1	9	2	3

i = 1 j

8	4	1	6	9	2	3

i = 1 j

8	1	4	6	9	2	3

i = 1 j

1	8	4	6	9	2	3

i = 1 j

1	8	4	6	9	2	3

i = 2 j

1	2	8	4	6	9	3

i = 3 j

1	2	3	8	4	6	9

i = 4 j

1	2	3	4	8	6	9

i = 5 j

1	2	3	4	6	8	9

i = 6 j

1	2	3	4	6	8	9

i = 7

j

Bubble Sort

Alg.: BUBBLESORT(A)

for i ← 1 **to** length[A]

  do for j ← length[A] **downto** i + 1

    do if A[j] < A[j -1]

      then exchange A[j] ↔ A[j-1]

i   j

8	4	6	9	2	3	1

i = 1   j

Bubble-Sort Running Time

Alg.: BUBBLESORT(A)

for i ← 1 **to** length[A] $\quad c_1$

do for j ← length[A] **downto** i + 1 $\quad c_2$

Comparisons: ≈ $n^2/2$ **do if** $A[j] < A[j-1]$ $\quad c_3$

then exchange $A[j] \leftrightarrow A[j-1]$ $\quad c_4$

Exchanges: ≈ $n^2/2$

$$T(n) = c_1(n+1) + c_2 \sum_{i=1}^{n}(n-i) + c_3 \sum_{i=1}^{n}(n-i) + c_4 \sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_2 + c_4) \sum_{i=1}^{n}(n-i)$$

$$\sum_{i=1}^{n}(n-i)$$

Thus, $T(n) = \Theta(n^2)$

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted
- Disadvantage:
  - Running time depends only slightly on the amount of order in the file

Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort

8	4	6	9	2	3	1

Alg.: SELECTION-SORT(*A*)

```
n ← length[A]
for j ← 1 to n - 1
    do smallest ← j
        for i ← j + 1 to n
            do if A[i] < A[smallest]
                then smallest ← i
        exchange A[j] ↔ A[smallest]
```

Analysis of Selection Sort

Alg.: SELECTION-SORT*(A)*

	cost	times
$n \leftarrow length[A]$	c_1	1
for $j \leftarrow 1$ **to** $n - 1$	c_2	n
do $smallest \leftarrow j$	c_3	n-1
for $i \leftarrow j + 1$ **to** n	c_4	$T(n)=c_1+c_2n+c_3(n-1)+c_4\sum_{j=1}^{n-1}(n-j+1)+c_5\sum_{j=1}^{n-1}(n-j)+c_6\sum_{j=2}^{n-1}(n-j)+c_7(n-1)=\Theta(n^2)$
do if $A[i] < A[smallest]$	c_5	$T(n)=c_1+c_2n+c_3(n-1)+c_4\sum_{j=1}^{n-1}(n-j+1)+c_5\sum_{j=1}^{n-1}(n-j)+c_6\sum_{j=2}^{n-1}(n-j)+c_7(n-1)=\Theta(n^2)$
then $smallest \leftarrow i$	c_6	$T(n)=c_1+c_2n+c_3(n-1)+c_4\sum_{j=1}^{n-1}(n-j+1)+c_5\sum_{j=1}^{n-1}(n-j)+c_6\sum_{j=2}^{n-1}(n-j)+c_7(n-1)=\Theta(n^2)$
exchange $A[j] \leftrightarrow A[smallest]$	c_7	n-1

$\approx n^2/2$ comparisons

$\approx n$ exchanges

$$T(n)=c_1+c_2n+c_3(n-1)+c_4\sum_{j=1}^{n-1}(n-j+1)+c_5\sum_{j=1}^{n-1}(n-j)+c_6\sum_{j=2}^{n-1}(n-j)+c_7(n-1)=\Theta(n^2)$$

MINIMUM AND MAXIMUM

```
MINIMUM(A)
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
return min
```