Sorting

Topics covered:

Selection Sort

Merge Sort

Quick Sort

Sorting:

Input: Sequence A[1...n]
Output: Permutation A'[1...n] of A[1...n] in non-decreasing order.

Selection Sort:

ex |8|4|2|5|2|

- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

```
* SelectionSort(A[1...n])
  for i from 1 to n:
      minIndex ← i
      for j from i+1 to n:
          if A[j] < A[minIndex]:
              minIndex ← j
      {A[minIndex] = min A[i...n]}
      Swap(A[i], A[minIndex])
      {A[1...i] is in final position}
```

Θ(n²)
running time
O(n²)
* requires a constant amount of extra memory.

Merge Sort:

ex |7|2|5|3|7|13|1|6|

Split the array in two half

|7|2|5|3| |7|13|1|6|

Sort the halves recursively.

Pseudo Code:

```
if n=1:
    return A
m ← ⌊n/2⌋
B ← MergeSort(A[1...m])
C ← MergeSort(A[m+1...n])
A' ← Merge(B,C)
return A'
```

Merging Two Sorted Arrays:

{ B and C are sorted }
D ← empty array of size p+q
while B and C are both non-empty:
  b ← the first element of B
  c ← the first element of C
  if b ≤ c:
    move b from B to the end of D
  else:
    move c from C to the end of D
move the rest of B and C to the end of D
return D

Running Time
= O(n log n)

Lemma:
$\log_2(n!) = \Omega(n \log n)$

Lower Bound for Comparision Based Sorting.

Lemma:
Any comparision based sorting algorithm perform $\Omega(n \log n)$ comparision in the worst case to sort n-objects

Non-comparision based Sorting Algorithms

1	2	3	4	5	6	7	8	9	10	11	12
2	3	2	1	3	2	2	3	2	2	2	1

1	2	3
1	2	3

event

we have sorted these number without actual comparising them

A':

1	2	2	2	2	2	2	2	3	3	3

Pseudo Code:

```
CountSort(A[1...n])
Count[1...m] ← [0,...0]
for i from 1 to n:
    Count[A[i]] ← Count[A[i]] + 1
{k appears count[k] times in A}
Pos[1...m] ← [0...0]
Pos[1] ← 1
for j from 2 to m:
    Pos[j] ← Pos[j-1] + Count[j-1]
{k will occupy range [Pos[k] .. Pos[k+1]-1]}
```

Running Time:
O(n)

```
For i from 1 to n:
    A'[Pos[A[i]]] ← A[i]
    Pos[A[i]] ← Pos[A[i]] + 1
```

Quick Sort:

- Comparision based algorithm
- Running time = $O(n \log n)$ (on average)
- effient in practice.

ex

| 6 | 4 | 8 | 2 | 9 | 3 | 9 | 4 | 7 | 6 | 1 |

partition with respect to $x = A[1]$

in particular, x is in its final position

| 1 | 4 | 2 | 3 | 4 | 6 | 6 | 9 | 7 | 8 | 9 |

≤ 6 $\quad$ > 6

Sort the two part recursively.

| 1 | 2 | 3 | 4 | 4 | 6 | 6 | 7 | 8 | 9 | 9 |

Algorithm: QuickSort(A, l, r)

```
if l ≥ r
    return
m ← Partition (A, l, r)
{A[m] is in the final position}
QuickSort (A, l, m-1)
QuickSort (A, m+1, r)
```

Partitioning: example:

- the pivot is $x = A[l]$
- move i from $l+1$ to r maintaing the following invalient
  - $A[k] \leq x$ for all $l+1 \leq k \leq j$
  - $A[k] \geq x$ for all $j+1 \leq k \leq i$

| 6 | 4 | 2 | 3 | 9 | 8 | 9 | 4 | 7 | 6 | 1 |

l $\quad$ j $\quad$ i

Pseudocode:

```
x ← A[l]   {pivot}
j ← l
for i from l+1 to r:
    if A[i] ≤ x:
        j ← j+1
        swap A[j] and A[i]
{A[l+1...j] ≤ x, A[j+1...i] > x}
swap A[l] and A[j]
return j
```

Date
Page

* Random Pivot:

$T(n) = O(n \log n)$

⇒ Unbalanced Partition:

- $T(n) = n + T(n-1)$:

  $T(n) = n + (n-1) + (n-2) + \dots = \Theta(n^2)$

- $T(n) = n + T(n-5) + T(4)$:

  $T(n) \geq n + (n-5) + (n-10) + \dots = \Theta(n^2)$

⇒ Balanced Partition:

- $T(n) = 2T(n/2) + n$:

  $T(n) = \Theta(n \log n)$

- $T(n) = T(n/10) + T(9n/10) + n$:

  $T(n) = \Theta(n \log n)$

⇒ Pseudo code:

```
Randomized QuickSort (A, l, r)
if l ≥ r:
    return
k ← random number between l and r  swap A[l] and A[k]
m ← Partition (A, l, r)
{A[m] is in the final position}
Randomized QuickSort (A, l, m-1)
Randomized QuickSort (A, m+1, r)
```

Theorem: Assume that all the elements of $A[1 \dots n]$ are pairwise different. Then the average running time of RandomQuickSort(A) is $O(n \log n)$ while the worst case running time is $O(n^2)$

* Equal Elements:

- What if all the elements of the given array are equal to each other?
- quick sort visualization
- The array is always split into two parts of size 0 and $n-1$
- $T(n) = n + T(n-1) + T(0)$ and hence $T(n) = \Theta(n^2)$!