1 Median/Order Statistics and Selection (CLRS §9.2, 9.3)

We saw in Quicksort the need for median.

More general question:

Input: A set A of n distinct numbers

Output:

Special cases: $i = 1$, $i = n$

1.1 A randomized algorithm (§9.2)

```
Choose j randomly from
k =
If k = i
    Return
Elseif k > i
    Return
Else
    Return
```

Example

1.1.1 Running time:

1.2 A deterministic algorithm (§9.3)

1. Partition A into

2. Compute median of

3. Compute median of

1.2.1 Running time:

2 Lower bounds on comparison queries

All the sorting/selecting algorithms we have seen so far are comparison based. They don't care about the actual numbers in the array, only their relative order. Comparison model: input items are "black boxes" and only allowed comparison. Time cost:

2.1 Example: Maximum finding algorithm (§9.1)

```
current_max =
For i = 3 to n
    current_max =
Return
```

Total number of comparisons:

Can we compute maximum with fewer comparisons?

Claim. Computing maximum of n elements requires
Proof.

2.2 Sorting (§8.1)

2.2.1 Decision Tree

Comparison algorithm can be viewed as tree of all possible comparisons, their outcomes, and the resulting answer.

Decision tree	Algorithm
internal node	
leaf	
	single execution of algorithm
	running time
	worst case running time

What is the number of comparisons performed on the worst case input?

2.2.2 Sorting lower bound

Corollary. MergeSort, QuickSort (with median pivot), are

3 Non-comparison based sorts

3.1 Counting sort

Best sorting algorithm is

Conjectured:

Example

Algorithm works, but we need to preserve the data associated with each key, not just sort the keys themselves.

Assume keys are in

```
L = array of
for j in range(n):
output = [ ]
for i in range(k):
    output.
```

CLRS:

```
For i = 1 to k
    C[i] = 0
For j = 0 to n
    C[A[j]] += 1
// C[i] now
For i = 2 to k
    C[i] += C[i-1]
//  C[i] now
For j = n downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] -= 1
```

Runtime:

3.2 Radix sort (§8.3)

Assume keys are in

4 Dynamic Programming! (CLRS Chapter 15, KT Chapter 6)

4.1 Intro and Memoized Fibonacci

Naive recursive algorithm:

```
if n <= 2: f = 1
else: f =
return f
```

Correct, but

Recurrence for running time

Memoized DP algorithm - the idea

4.2 Rod cutting (CLRS §15.1)

What is the highest revenue we can get by cutting an n foot rod and selling the pieces?

Example

Q: How many total ways are there to cut the rod?

Strategy: Look at a single step and reduce to a previous problem.

Naive recursive algorithm:

```
if n == 0:
    return
q =
for i = 1 to n:
    q =
return q
```

Running time

Memoized DP algorithm

```
if memo[n] exists:
    return




memo[n] =
return
```

Running time

Bottom-up version

```
r[0] = 0
for j = 1 to n
    q =
    for i = 1 to
        q =
    r[j] =
return
```

How do we output the actual sequence of cuts we should do?

```
r[0] = 0
for j = 1 to n
    q =
    for i = 1 to
        if q
            q =
            s[j] =
    r[j] =
print r[n]
while n > 0
    print
    n =
```

4.3 Longest common subsequence (CLRS §15.4)

Example application: bioinformatics (similarity between DNA sequences)

Input:

Output:

Example

Optimal substructure:

Theorem 1. *If $X = x_1 \dots x_m$, $Y = y_1 \dots y_n$, and $Z = z_1 \dots z_k$ is an LCS of X and Y,*

1. *If $x_m = y_n$, then*

2. *If $x_m \neq y_n$, then*

   (a) *if $z_k \neq x_m$, then*

   (b) *if $z_k \neq y_m$, then*

- Subproblems:

- Recurrence:

- Naive recursion vs memoized:

Bottom up algorithm:

```
for i = 1 to m:
    C[i, 0] =
for j = 1 to n:
    C[0, j] =
for i = 1 to m:
    for j = 0 to n:
        if
            C[i, j] =
            B[i, j] =
        else if
            C[i, j] =
            B[i, j] =
        else
            C[i, j] =
            B[i, j] =
return
```