Find Min and Max

Find them independantly: $2n-2$.

- Can easily modify to get $2n-3$.

Should be able to do better(?)

Try divide and conquer.

```
Find_Max_Min(ELEM *L, int lower, int upper) {
  if (upper == lower) return lower, lower;    // n=1
  if (upper == lower+1)                       // n=2
    return max(L[upper], L[lower]),
           min(L[upper], L[lower]); // Only 1 compare
  mid = (lower + upper)/2;                    // n>2
  max1, min1 = Find_Max_Min(L, lower, mid);
  max2, min2 = Find_Max_Min(L, mid+1, upper);
  return max(L[max1], L[max2]), min(L[min1], L[min2]);
}
```

Recurrence:

$$f(n) = \begin{cases} 2f(n/2) + 2 & n > 2 \\ 1 & n = 2 \end{cases}$$

Solving the Recurrence

Assume $n = 2^k$.
Let's expand the recurrence a bit.

$$
\begin{aligned}
f(n) &= 2f(n/2) + 2 \\
&= 2[2f(n/4) + 2] + 2 \\
&= 4f(n/4) + 4 + 2 \\
&= 4[2f(n/8) + 2] + 4 + 2 \\
&= 8f(n/8) + 8 + 4 + 2 \\
&= 2^i f(n/2^i) + \sum_{j=1}^{i} 2^j \\
&= 2^{k-1} f(n/2^{k-1}) + \sum_{j=1}^{k-1} 2^j \\
&= 2^{k-1} f(2) + \sum_{j=1}^{k-1} 2^j \\
&= 2^{k-1} + \sum_{j=1}^{k-1} 2^j \\
&= n/2 + 2^k - 2 \\
&= 3n/2 - 2
\end{aligned}
$$

Looking Closer

But its not always true that $n = 2^k$.

The true cost recurrence is:

$$f(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + 2 & n > 2 \end{cases}$$

Here is what really happens:

n	2	3	4	5	6	7	8	9	10	11
$f(n)$	1	2	4	6	8	9	10	12	14	16
$3n/2 - 2$	1	2.5	4	5.5	7	8.5	10	11.5	13	14.5

The true cost for $f(n)$ ranges between $3n/2 - 2$ and $5n/3 - 2$.

- For what sort of input does the algorithm work best?

Finding a Better Algorithm

What is the cost with six values?

What if we divide into a group of 4 and a group of 2?

With divide and conquer, we seek to minimize the work, not necessarily balance the input sizes.

When does the algorithm do its best?

What about 12? 24?

Lesson: For divide and conquer, pay attention to what happens for small n.

Algorithms from Recurrences

What does this model?

$$f(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ \min_{1 \le k \le n-1} \{ f(k) + f(n-k) \} + 2 & n > 2 \end{cases}$$

n	1	2	3	4	5	6	7	8
3	$\underline{3}$	$\underline{3}$						
4	5	$\underline{4}$	5					
5	7	$\underline{6}$	$\underline{6}$	7				
6	9	$\underline{7}$	8	$\underline{7}$	9			
7	11	$\underline{9}$	$\underline{9}$	$\underline{9}$	$\underline{9}$	11		
8	13	$\underline{10}$	11	$\underline{10}$	11	$\underline{10}$		13
9	15	$\underline{12}$	$\underline{12}$	$\underline{12}$	$\underline{12}$	$\underline{12}$	$\underline{12}$	15

$k = 2$ looks promising.

$$f(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ f(2) + f(n-2) + 2 & n > 2 \end{cases}$$

Cost:

What is the corresponding algorithm?

The Lower Bound

Is $\lceil 3n/2 \rceil - 2$ optimal?

Consider all states that a successful algorithm must go through: The **state space** lower bound.

At any given instant, track the following four categories:

- Novices: not tested.
- Winners: Won at least once, never lost.
- Losers: Lost at least once, never won.
- Moderates: Both won and lost at least once.

Who can get ignored?

What is the initial state?

What is the final state?

How is this relevant?

Lower Bound (cont.)

Every algorithm must go from $(n, 0, 0, 0)$ to $(0, 1, 1, n-2)$.

There are 10 types of comparison.

Comparing with a moderate cannot be more efficient than other comparisons, so ignore them.

If we are in state (i, j, k, l) and we have a comparison, then:

$$
\begin{array}{lllll}
N : N & (i-2, & j+1, & k+1, & l) \\
W : W & (i, & j-1, & k, & l+1) \\
L : L & (i, & j, & k-1, & l+1) \\
L : N & (i-1, & j+1, & k, & l) \\
\quad or & (i-1, & j, & k, & l+1) \\
W : N & (i-1, & j, & k+1, & l) \\
\quad or & (i-1, & j, & k, & l+1) \\
W : L & (i, & j, & k, & l) \\
\quad or & (i, & j-1, & k-1, & l+2)
\end{array}
$$

Adversarial Argument

What should an adversary do?

- Comparing a winner to a loser is of no value.

Only the following five transitions are of interest:

$N:N$	$(i-2,$	$j+1,$	$k+1,$	$l)$
$L:N$	$(i-1,$	$j+1,$	$k,$	$l)$
$W:N$	$(i-1,$	$j,$	$k+1,$	$l)$
$W:W$	$(i,$	$j-1,$	$k,$	$l+1)$
$L:L$	$(i,$	$j,$	$k-1,$	$l+1)$

Only the last two types increase the number of moderates, so there must be $n-2$ of these.

The number of novices must go to 0, and the first is the most efficient way to do this: $\lceil n/2 \rceil$ are required.

Finding the ith Best

We need to find the following poset:

We don't care about the relative order within the upper and lower groups.

Can we do better than sorting? ($\Theta(n \log n)$)

Can we tighten the lower bound beyond n?

What if we want to find the median element?

Splitting a List

Given an arbitrary element, split the list into those elements less and those elements greater.

```
int Split(ELEM *L, int lower, int upper, int piv_loc) {
  ELEM pivot = L[piv_loc];
  swap(L[lower], L[piv_loc]);
  piv_loc = lower;
  for (i=lower+1; i<=upper; i++)
    if (pivot > L[i]) {
      piv_loc++;
      swap(L[i], L[piv_loc]);
    }
  swap(L[lower], L[piv_loc]);
  return piv_loc;
}
```

If the pivot is ith best, we are done.

If not, solve the subproblem recursively.

Cost

What is the worst case cost of this algorithm?
Under what circumstances?
What is the average case cost if we pick the pivots at random?

Let $f(n,i)$ be the average time to find the ith best of n elements.

$$\begin{aligned} f(n,i) &= n-1+\frac{1}{n}\sum_{k=1}^{n-i} f(n-k,i)+\frac{1}{n}0 \\ &\quad +\frac{1}{n}\sum_{k=n-i+2}^{n} f(k-1,i+k-n-1). \end{aligned}$$

Set $j = n-k+1$.

$$\begin{aligned} f(n,i) &= n-1+\frac{1}{n}\sum_{j=i+1}^{n} f(j-1,i) \\ &\quad +\frac{1}{n}\sum_{j=1}^{i-1} f(n-j,i-j). \end{aligned}$$

Let $f(n)$ be the cost averaged over all i.

$$f(n) = \frac{1}{n}\sum_{i=1}^{n} f(n,i).$$

Technique

$$
\begin{aligned}
nf(n) &= \sum_{i=1}^{n} f(n,i) \\
&= n^2 - n + \frac{1}{n}\sum_{i=1}^{n}\left\{\sum_{j=i+1}^{n} f(j-1,i) + \right. \\
&\qquad \left. \sum_{j=1}^{i-1} f(n-j,i-j)\right\}.
\end{aligned}
$$

It turns out that the two double sums are the same (just going from different directions).

$$
\begin{aligned}
nf(n) &= n^2 - n + \frac{2}{n}\sum_{j=1}^{n-1}\sum_{i=1}^{j} f(j,i) \\
&= n^2 - n + \frac{2}{n}\sum_{j=1}^{n-1} jf(j)
\end{aligned}
$$

Technique (cont.)

Therefore,

$$n^2 f(n) = n^3 - n^2 + 2 \sum_{j=1}^{n-1} j f(j).$$

This is an example of a **full history** recurrence.

Solving the Recurrence

If we subtract the appropriate form of $f(n-1)$, most of the terms will cancel out.

$$
\begin{aligned}
n^2 f(n) - (n-1)^2 f(n-1) & \\
&= n^3 - n^2 + 2\sum_{j=1}^{n-1} jf(j) \\
&\quad -(n-1)^3 + (n-1)^2 - 2\sum_{j=1}^{n-2} jf(j) \\
&= 3n^2 - 5n + 2 + 2(n-1)f(n-1) \\
\Rightarrow n^2 f(n) &= (n^2-1)f(n-1) + 3n^2 - 5n + 2.
\end{aligned}
$$

Estimate:

$$
\begin{aligned}
n^2 f(n) &= (n^2-1)f(n-1) + 3n^2 - 5n + 2 \\
&< n^2 f(n-1) + 3n^2 \\
\Rightarrow f(n) &< f(n-1) + 3 \\
\Rightarrow f(n) &< 3n
\end{aligned}
$$

Therefore, $f(n)$ is in $O(n)$.

Does this mean that the worst case is linear?

Improving the Worst Case

Want worst case linear algorithm.

Goal: Pick a pivot that guarentees discarding a fixed proportion of the elements.

Can't just choose a pivot at random.

Median would be ideal – too expensive.

Choose a constant c, pick the median of a sample of size n/c elements.

Will discard at least $n/2c$ elements.

Selecting an Approximate Median

Algorithm:

- Choose the $n/5$ medians for groups of 5 elements of L.
- Recursively, select the median of the $n/5$ elements.
- Use SPLIT to partition the list into large and small elements around the "median."

Now, the algorithm for finding the ith element uses the median finding algorithm to recursively reach the goal.

Constructive Induction

Is the following recurrence linear?

$$f(n) \leq f(\lceil n/5 \rceil) + f(\lceil (7n-5)/10 \rceil) + 6\lceil n/5 \rceil + n - 1.$$

To answer this, assume it is true for some constant r such that $f(n) \leq rn$ for all n greater than some bound.

$$\begin{aligned}
f(n) &\leq f(\lceil \frac{n}{5} \rceil) + f(\lceil \frac{7n-5}{10} \rceil) + 6\lceil \frac{n}{5} \rceil + n - 1 \\
&\leq r(\frac{n}{5} + 1) + r(\frac{7n-5}{10} + 1) + 6(\frac{n}{5} + 1) + n - 1 \\
&\leq (\frac{r}{5} + \frac{7r}{10} + \frac{11}{5})n + \frac{3r}{2} + 5 \\
&\leq \frac{9r+22}{10}n + \frac{3r+10}{2}.
\end{aligned}$$

This is true for $r \geq 23$ and $n \geq 380$.

Thus, we can use induction to prove that,

$$\forall n \geq 380, f(n) \leq 23n.$$

Actually, this algorithm is not practical.
Better to rely on "luck."