CS 4104: Data and Algorithm Analysis

Clifford A. Shaffer

Department of Computer Science
Virginia Tech
Blacksburg, Virginia

Fall 2010

Selection

How can we find the *i*th largest value

- in a sorted list?
- in an unsorted list?

Can we do better with an unsorted list than to sort it?

Assumption: Elements can be **ranked**.

Properties of Relationships (1)

Partial Order: Given a set S and a binary operator R, R defines a partial order on S if R is:

- Antisymmetric: Whenever aRb and bRa, then $a = b$, for all $a, b \in \mathbf{S}$.
- Transitive: Whenever aRb and bRc, then aRc, for all $a, b, c \in \mathbf{S}$.

Think of a relationship as a set of tuples.

- A tuple is in the set (in the relation) iff the relation holds on that tuple.

Example: S is Integers, R is $<$.

Example: S is the power set of $\{1, 2, 3\}$, R is subset.

Properties of Relationships (2)

A partial order is also called a **poset**.

If every pair of elements in S is relatable by R, then we have a **linear order**.

General Model

For all of our problems on Selection and Sorting:

- The poset has a linear ordering. (Usually natural numbers and a relationship of $\leq$.)
- Cost measure is the number of 3-way element-element comparisons.

Selection problems:

- Find the max or min.
- Find the second largest.
- Find the median.
- Find the *i*th largest.
- Find several ranks simultaneously.

Finding the Maximum

```
int Find_max(int *L, int low, int high) {
  max = low;
  for(i=low+1; i<= high; i++)
    if(L[i] > L[max])
      max = i;
  return max;
}
```

What is the cost?

Is this optimal?

Proof of Lower Bound (1)

Try #1:

- The winner must compare against all other elements, so there must be $n - 1$ comparisons.

Proof of Lower Bound (1)

Try #1:

- The winner must compare against all other elements, so there must be $n - 1$ comparisons.

Try #2:

- Only the winner does not lose.
- There are $n - 1$ losers.
- A single comparison generates (at most) one (new) loser.
- Therefore, there must be $n - 1$ comparisons.

Proof of Lower Bound (2)

Alternative proof:

- To find the max, we must build a poset having one max and $n - 1$ losers, starting from a poset of n singletons.
- We wish to connect the elements of the poset with the minimum number of links.
- This requires at least $n - 1$ links.
- A comparison provides at most one new link.

Average Cost

- What is the average cost for `Find_max`?
  - Since it always does the same number of comparisons, clearly $n - 1$ comparisons.
- How many assignments to `max` does it do?
- Ignoring the actual values in L, there are $n!$ permutations for the input.
- `Find_max` does an assignment on the ith iteration iff $L[i]$ is the biggest of the first i elements.
- Since this event does happen, or does not happen:
  - Given no information about distribution, the probability of an assignment after each comparison is 50%.

Average Number of Assignments

`Find_max` does an assignment on the ith iteration iff $L[i]$ is the biggest of the first i elements.

Assuming all permutations are equally likely, the probability of this being true is $1/i$.

$$1 + \sum_{i=2}^{n} \frac{1}{i} \times 1 = \sum_{i=1}^{n} \frac{1}{i}.$$

This sum generates the nth **harmonic number**: $\mathcal{H}_n$.

Technique (1)

Since $i \leq 2^{\lceil \log i \rceil}$, $1/i \geq 1/2^{\lceil \log i \rceil}$.

Thus, if $n = 2^k$

$$\begin{aligned}
\mathcal{H}_{2^k} &= 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{2^k} \\
&\geq 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} \\
&\qquad + \ldots + \frac{1}{2^k} \\
&= 1 + \frac{1}{2} + \frac{2}{4} + \frac{4}{8} + \ldots \frac{2^{k-1}}{2^k} \\
&= 1 + \frac{k}{2}.
\end{aligned}$$

Technique (2)

Using similar logic, $\mathcal{H}_{2^k} \leq k + \frac{1}{2^k}$. Thus, $\mathcal{H}_n = \Theta(\log n)$.

More exactly, $\mathcal{H}_n$ is close to $\ln n$.

Variance (1)

How “reliable” is the average?

- How much will a given run of the program deviate from the average?

Variance: For runs of the program, average square of differences.

Standard deviation: Square root of variance.

From Čebyšev’s Inequality, 75% of the observations fall within 2 standard deviations of the average.

For `Find_max`, the variance is

$$\mathcal{H}_n - \frac{\pi^2}{6} = \ln n - \frac{\pi^2}{6}$$

Variance (2)

The standard deviation is thus about $\sqrt{\ln n}$.

- So, 75% of the observations are between $\ln n - 2\sqrt{\ln n}$ and $\ln n + 2\sqrt{\ln n}$.
- Is this a narrow spread or a wide spread?

Finding the Second Best

In a single-elimination tournament, is the second best the one who loses in the finals? Simple algorithm:

- Find the best.
- Discard it.
- Now, find the second best of the $n - 1$ remaining elements.

Cost? Is this optimal?

Lower Bound for Second (1)

Lower bound:

- Anyone who lost to anyone who is not the max cannot be second.
- So, the only candidates are those who lost to max.
- `Find_max` might compare max to $n - 1$ others.
- Thus, we might need $n - 2$ additional comparisons to find second.
- Wrong!

Lower Bound for Second (2)

The previous argument exhibits the **necessity fallacy**:

- Our algorithm does something, therefore all algorithms solving the problem must do the same.

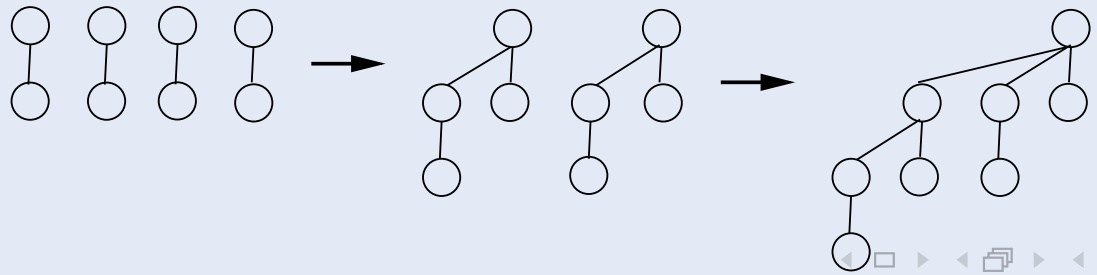
Alternative: Divide and conquer

- Break the list into two halves.
- Run `Find_max` on each half.
- Compare the winners.
- Run `Find_max` on the winner's half for second.
- Compare that second to second winner.

Cost: $\lceil 3n/2 \rceil - 2$.
Is this optimal?
What if we break the list into four pieces? Eight?

Binomial Trees (1)

- Pushing this idea to its extreme, we want each comparison to be between winners of equal numbers of comparisons.
- The only candidates for second are losers to the eventual winner.
- A **binomial tree** of height m has 2^m nodes organized as:
  - a single node, if $m = 0$, or
  - two height $m - 1$ binomial trees with one tree's root becoming a child of the other.



Binomial Trees (2)

Algorithm:

- Build the tree.
- Compare the $\lceil \log n \rceil$ children of the root for second.

Cost?

Binomial Tree Representation

- We could store the binomial tree as an explicit tree structure.
- Can also store binomial tree implicitly: In array.
- Assume two trees, each with 2^k nodes, are in array as:
  - First tree in positions 1 to 2^k.
  - Second tree in positions $2^k + 1$ to 2^{k+1}.
  - The root of a subtree is in the final array position for that subtree.
- To join:
  - Compare the roots of the subtrees.
  - If necessary, swap subtrees so larger root element is second subtree.
- Trades space for time.

Adversarial Lower Bounds Proof (1)

Many lower bounds proofs use the concept of an **adversary**.

The adversary's job is to make an algorithm's cost as high as possible.

The algorithm asks the adversary for information about the input.

The adversary may never lie.

Adversarial Lower Bounds Proof (2)

Imagine that the adversary keeps a list of all possible inputs.

- When the algorithm asks a question, the adversary answers, and crosses out all remaining inputs inconsistent with that answer.
- The adversary is permitted to give any answer that is consistent with at least one remaining input.

Examples:

- Hangman.
- Search an unordered list.

Lower Bound for Second Best

At least $n - 1$ values must lose at least once.

- At least $n - 1$ compares.

In addition, at least $k - 1$ values must lose to the second best.

- I.e., k direct losers to the winner must be compared.

There must be at least $n + k - 2$ comparisons.

How low can we make k?

Adversarial Lower Bound

Call the **strength** of element $L[i]$ the number of elements $L[i]$ is (known to be) bigger than.

If $L[i]$ has strength a, and $L[j]$ has strength b, then the winner has strength $a + b + 1$.

What should the adversary do?

- Minimize the rate at which any element improves.
- Do this by making the stronger element always win.
- Is this legal?

Lower Bound (Cont.)

What should the algorithm do?

If $a \geq b$, then $2a \geq a + b$.

- From the algorithm's point of view, the best outcome is that an element doubles in strength.
- This happens when $a = b$.
- All strengths begin at zero, so the winner must make at least k comparisons for $2^{k-1} < n \leq 2^k$.

Thus, there must be at least $n + \lceil \log n \rceil - 2$ comparisons.

Find Min and Max (1)

Find them independantly: $2n - 2$.

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

Should be able to do better(?)

Try divide and conquer.

Find Min and Max (2)

```
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]); // 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 (1)

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

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

Solving the Recurrence (2)

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

Looking Closer (1)

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 (1)

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.

Algorithms from Recurrences (1)

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

Cost: What is the corresponding algorithm?

The Lower Bound (1)

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.

The Lower Bound (2)

Who can get ignored?

What is the initial state?

What is the final state?

How is this relevant?

Lower Bound (3)

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.

Lower Bound (3)

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

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

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:

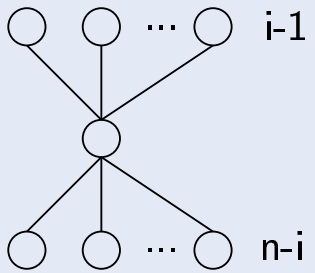
$N:N$	$(i-2,$	$j+1,$	$k+1,$	$l)$
$L:N$	$(i-1,$	$j+1,$	$k,$	$l)$
$W:N$	$(i-1,$	$j,$	$k+1,$	$l)$
$W:W$	$(i,$	$j-1,$	$k,$	$l+1)$
$L:L$	$(i,$	$j,$	$k-1,$	$l+1)$

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 *i*th 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, partition the list into those elements less and those elements greater.

```
// Initially, l and r are one position to left and
//   right of the subarray, respectively
int partition(Elem A[], int l, int r, Elem pivot) {
  do {                  // Move bounds inward to meet
    while (A[++l] < pivot); // Move l right and
    while ((l < r) && (A[--r] > pivot)); // r left
    swap(A, l, r);              // Swap values
  } while (l < r);              // Stop when they cross
  return l;       // Return first position on right
}
```

If the pivot position is *i*th best, we are done.
If not, solve the subproblem recursively.

Cost (1)

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

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

- Array bounds go from 1 to n
- Call j the position of the pivot

$$f(n,i) = (n-1) + \frac{1}{n}\sum_{j=i+1}^{n} f(j-1,i) + \frac{1}{n}0 + \frac{1}{n}\sum_{j=1}^{i-1} f(n-j,i-j).$$

Cost (2)

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

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

Note: Even if we just want to analyze for median-finding, still need to be able to solve for arbitrary i on recursive calls.

Technique (1)

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

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

Technique (2)

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

Therefore,

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

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

Solving the Recurrence (1)

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

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

Solving the Recurrence (2)

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.

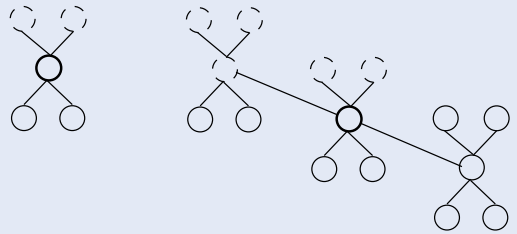
Choose a constant c, pick the median of a sample of size n/c elements.

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.”



- For 5, discard at least 2
- For 15, discard at least 5
- For 25, discard at least 8
- In general, discard at least $(3n + 5)/10$

Constructive Induction (1)

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.

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

Constructive Induction (2)

Try $r = 1$: $3.1n + 7.5 \leq n$ which doesn't work.
Try $r = 23$: Get $22.9n + 39.5 \leq 23n$.
This is true for $n \geq 395$.

Thus, we can use induction to prove that,

$$\forall n \geq 395, f(n) \leq 23n.$$

This algorithm is not practical. Better to rely on "luck."