Recall selection problem from lecture 18

\[
\begin{array}{l}
\text{select (list, i)} \\
\quad \text{choose pivot} \\
\quad \text{partition the list around pivot} \\
\quad l_1 \text{ is the list of elements } \leq \text{ pivot} \\
\quad l_2 \text{ is } \cdots \cdots > \text{ pivot} \\
\quad \text{if } (i = l_1.\text{size}) \\
\quad \quad \text{return pivot} \\
\quad \text{else if } (i < l_1.\text{size}) \\
\quad \quad \text{return select (l_1, i)} \\
\quad \text{else return select (l_2, i - (l_1.\text{size} + 1))} \\
\end{array}
\]

What if we choose pivot \( p \) uniformly at random
is. equal probability for each list position?

Suppose the list has \( n \) elements.

\[
\begin{array}{c|c}
\text{list.size} & \text{size} \\
\hline
0 & n \\
1 & n - 1 \\
2 & n - 2 \\
\vdots & \vdots \\
\end{array}
\]

Consider recursive calls \( \text{select (list, i)} \)

We look at how list.size decreases.

\[Q: \text{What is the probability that list.size is decreased by some factor e.g. list.size} \leftarrow \text{list.size} \times \frac{3}{4} \text{ on any call of select (list, i)?} \]

There are two issues here:

1. whether a good pivot is chosen i.e. how balanced is the \( l_1, l_2 \) split.
2. whether the \( i \)-th element is in \( l_1 \) or \( l_2 \)

It is relatively difficult to disentangle these.

• worst case:
  list.size decreases by 1
• best case: (assuming \( i \)-th element not found)
  list.size decreases to 1

If we choose the pivot randomly, then
list.size will decrease randomly.

(Continued on next page)
For example, it is possible to choose a bad pivot (l₁ and l₂ are unbalanced) and yet still shrink list size by alot.

e.g. Select \((l₁, 0)\)

\[ \text{list} \]

We got lucky here \((i = 0)\).

To simplify the probability calculation, we pose a slightly different question.

Both of these conditions are met:

\[ \frac{n}{4} \leq \text{pivot position} < \frac{3}{4}n \]

We call this a good choice of pivot.

\[ \frac{n}{4} \quad \frac{3}{4}n \]

This condition holds with probability \(\frac{1}{2}\)

(which is why we chose \(\frac{3}{4}\) as the factor).

Q: What is the probability that \(l₁, \text{size} \) and \(l₂, \text{size} \) are both less than \(\frac{3}{4}n\)?

A:

\[ l₁, \text{size} < \frac{3}{4}n \iff \text{pivot position} < \frac{3}{4}n \]

\[ l₂, \text{size} < \frac{3}{4}n \iff \frac{n}{4} \leq \text{pivot position} \]

Summary so far:

With probability \(p > \frac{1}{2}\), a randomly selected pivot will result in the list in the next recursive select call being at most \(\frac{3}{4}\) as large as the current list.

(Why \(p > \frac{1}{2}\) and not \(p = \frac{1}{2}\) ? Because a bad pivot might still yield a small list in the next select call.)

Partition the real line interval \((0, n]\) into disjoint intervals:

\[ \text{interval } j = \left( \left( \frac{3}{4} \right)^{j+1} n, \left( \frac{3}{4} \right)^j n \right] \]

\[ \text{etc. } 4 \quad 3 \quad 2 \quad 1 \quad 0 \]

\[ 0 \quad \text{etc. } (\frac{3}{4})^n \quad \frac{3}{4}n \quad n \]

ASIDE: We only need \(j \leq \log_{\frac{3}{4}} n\), since we will only care about interval sizes \(\geq 1\).
We say the algorithm is in phase $j$ when it makes a `Select(list,i)` call with list.size in interval $j$, i.e., it starts in phase $j=0$.

```
etc  j=3  j=2  j=1  j=0
0  etc  etc (3/4)n  3/4 n  n
```

Let random variable $X_j$ be the number of times that `select` is called recursively when list.size in interval $j$. The algorithm is in phase $j$.

```
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
0 0 1 0 1 2 0 2
```

Let $t(n)$ be the time taken by `select` for randomly chosen pivots.

\[ t(n) \leq \sum_{j=0}^{\infty} X_j \cdot C \left( \frac{3}{4} \right)^j n \]

\text{time needed to partition a list of size } \left( \frac{3}{4} \right)^j n

With probability $p > \frac{1}{2}$, a randomly selected pivot will result in the next `select` call being on a list that is at most $\frac{3}{4}$ as large; and hence in a different phase, i.e., list.size in a different interval.\[\]

Now take expected values and use linearity of expectation.

\[ E(t(n)) \leq \sum_{j=0}^{\infty} E(X_j) \cdot C \left( \frac{3}{4} \right)^j \]

\[ \text{how to calculate?} \]

\[ E(X_j) = \text{expected number of recursive } \]

\[ \text{select calls in phase } j \]

\[ \leq \text{expected number of times you flip a coin } (p=\frac{1}{2}) \]

\[ \text{until you get heads} \]

\[ \text{i.e., heads } \Rightarrow \text{jump to next phase} \]

\[ = 2 \quad (\text{from last lecture}) \]
\[ E_t(n) \leq \sum_{j=0}^{\infty} E[X_j] \cdot C \left( \frac{3}{4} \right)^j n \]
\[ < 2Cn \sum_{j=0}^{\infty} \left( \frac{3}{4} \right)^j \]
\[ = 2Cn \frac{1}{1 - \frac{3}{4}} \]
\[ = 8Cn \]

Thus, \( E_t(n) \) is \( O(n) \).

**Summary of Main Idea**

- Each recursive call to \texttt{select} shrinks the list by a constant factor \( (\frac{3}{4}) \) with probability \( p > \frac{1}{2} \). Thus, the expected number of recursive calls to shrink by that constant factor is \( < 2 \).
- Partitioning a list takes linear time i.e. \( c \cdot \text{list size} \)
- Thus, expected time is at most \( 2Cn(1 + r + r^2 + \ldots) \) where \( r = \frac{3}{4} \), which is \( O(n) \).

**Lecture 20**

**Randomized Algorithms**

**Expected Run Time Analysis**

- \texttt{quickselect}

  - (I used Kleinberg & Tardos)

  \[ \text{quicksort}(\text{list}) \]

  \[ \text{if} \ (\text{list size}) \leq 1 \]

  \[ \text{return list} \]

  \[ \text{else} \]

  \[ \text{choose random pivot (uniform)} \]

  \[ \text{l1} \leftarrow \text{elements less than pivot} \]

  \[ \text{l2} \leftarrow \text{elements greater than pivot} \]

  \[ \text{l1} \leftarrow \text{quicksort}(\text{l1}) \]

  \[ \text{l2} \leftarrow \text{quicksort}(\text{l2}) \]

  \[ \text{return concatenate} (\text{l1}, \text{pivot}, \text{l2}) \]

To simplify the analysis, let's suppose we make recursive calls only when we have found a good pivot, namely \( l1.\text{size} \) and \( l2.\text{size} \) are both at least \( \frac{n}{4} \) (or equivalently, both at most \( \frac{3n}{4} \)).

As we saw earlier, a good pivot is chosen with probability \( p = \frac{1}{2} \).

\[ \begin{array}{ccccccc}
0 & \frac{n}{4} & \frac{n}{2} & \frac{3n}{4} & \frac{3n}{2} & \ldots
\end{array} \]
Q: What is the probability that the body of the while loop is executed infinitely many times, i.e., infinite loop?

A: \( \left( \frac{1}{2} \right)^i \to 0 \) as \( i \to \infty \).

Let \( Y_j \) be the number of subproblems of quicksort such that

\[
\text{list size is in} \quad \left( \left( \frac{3}{4} \right)^j n, \left( \frac{3}{4} \right)^{j+1} n \right].
\]

We will call these problems of type \( j \) and soon we will calculate \( \mathbb{E}[Y_j] \).

Total work for quicksort

\[
\leq \sum_{j} \left( \text{work for each subproblem of type } j \right) \cdot Y_j
\]

Now take expected value

\[\mathbb{E}(\text{total work for quicksort}) = ?\]

A few observations...

Each node in the quicksort call tree has two children whose sizes are at most \( \frac{3}{4} \) as large. Thus,

- the height of the quicksort call tree is \( \log_{\frac{3}{4}} n \).
- parent and child contribute to different \( Y_j \) i.e., they are different type of subproblems.

Two nodes in the quicksort call tree have overlapping (not disjoint) lists if and only if one is an ancestor of the other (parent-child, grandparent-grandchild, etc). Thus, all subproblems of type \( j \) are disjoint.
**MODIFIED April 10**

Q: How big is $Y_j$?
A: Since the size of each subproblem of type $j$ at least $(\frac{3}{4})^{j+1}n$ and since subproblems of type $j$ don't overlap, we have $Y_j \times (\frac{3}{4})^{j+1}n \leq n$. (see next slide)

Thus, $Y_j \leq \left(\frac{4}{3}\right)^{j+1}$.

---

**ASIDE:** [added April 10]

We have $Y_j$ subproblems of type $j$ and let's say that they are of sizes $l_1, l_2, ..., l_{Y_j}$. Then $\sum_{i=1}^{Y_j} l_i \leq n$.

But $(\frac{3}{4})^{j+1}n \leq l_i$ for each $i = 1, ..., Y_j$.

Substituting $l_i$ gives:

$$(\frac{3}{4})^{j+1}n \cdot Y_j \leq n.$$  

**Summary of main idea**

- Quick sort recursively divides into subproblems $j$ we can upper bound the number of problems of a given size. This upper bound on number grows as the problem size shrinks; the two effects cancel, leaving a total linear work (proportional to $n$) for each problem size, and the number of problem sizes is $\log n$.

---

**Announcements**

- Next week is the last 2 lectures (The following week I will hold open office hours 9-5)

---

**Final Exam**

**COMP 351 Algorithms and Data Structures**

**Tuesday April 15, 2014 9 AM**

**Examiner:** Michael Langer  
**Associate Examiner:** Joseph Vishal

**Instructions:**  
- This is a closed-book exam.  
- You may use up to five double-sided CRB sheets.  
- No electronic devices are allowed.  
- Your answer sheet will be graded page by page, then the reverse side and indicate that you have done so.

<table>
<thead>
<tr>
<th>Questions</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

- Midterm 1 (15)
- Midterm 2 (15)
- After midterm 2 (20)