Selection Problems

Medians and Order Statistics
Order Statistics

- The $i^{th}$ order statistic of a set of $n$ numbers is the $i^{th}$ smallest element in sorted sequence:

  \[ A \]
  
  \[
  \begin{array}{cccccc}
  4 & 1 & 3 & 2 & 16 & 9 & 10 & 14 & 8 & 7 \\
  \end{array}
  \]

- Minimum or first order statistic: 1
- Maximum or $n^{th}$ order statistic: 16
- Median or $(n/2)^{th}$ order statistic: 7 or 8
  (both are medians, happens when $n$ is even!)

The Selection problem:

- Input: An array $A$ of distinct numbers of size $n$, and a number $i$.
- Output: The element $x$ in $A$ that is larger than exactly $i-1$ other elements in $A$.

Finding maximum and minimum can be easily solved in linear time ($O(n)$).
(it’s actually $\Theta(n)$).
Trivial Solution:

- Sort the array $A$, and return the entry in $i^{th}$ position:
  - Sorting $A$ takes $O(n \log n)$.
  - The $i^{th}$ entry can be returned in constant time.
- Worst case running time: $O(n \log n)$
- Can we do better?
  Comparing to maximum and minimum, the general $i$ is taking a long time.

A Randomized Selection Algorithm (idea):

- Think about the properties of $\text{Partition}( )$ algorithm:

<table>
<thead>
<tr>
<th>&lt;= $x$</th>
<th>$x$</th>
<th>&gt;$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$q$</td>
<td>$r$</td>
</tr>
</tbody>
</table>

- If $i=q$, then we have $x$ as the $i^{th}$ order statistic.
  (what if this not the case?)
A Randomized Selection Algorithm (idea):

- If \( i < q \), then we have look for the \( i \)th order statistic among first \( p-q+1 \) elements:

  \[
  \begin{array}{c}
  i \\
  \hline
  \leq x & x & \geq x \\
  \hline
  p & q & r
  \end{array}
  \]

- We can call \textbf{Partition( )}, with parameters \((A,p,q)\)

A Randomized Selection Algorithm (idea):

- If \( i > q \), then we have look for \( i \)th order statistic among elements between \( q \) and \( r \):

  \[
  \begin{array}{c}
  i \\
  \hline
  \leq x & x & \geq x \\
  \hline
  p & q & r
  \end{array}
  \]

- We can call \textbf{Partition( )}, with parameters \((A,q,r)\)
The Algorithm:

Randomized-Select($A, p, r, i$)
    if $p = r$ then
        Return $A[p]$
    $q =$ Randomized-Partition($A, p, r$)
    $k = q - p + 1$
    if $i \leq k$ then
        Randomized-Select($A, p, q, i$)
    else
        Randomized-Select($A, q, r, i - k$)

Running time:

- The recurrence:
  - Lucky: $T(n) = T(9n/10) + \Theta(n) = \Theta(n)$
    Using master theorem:
    
    $\log_{10} 1
     \frac{n}{7} = n^0 = 1$

  - Unlucky: $T(n) = T(n - 1) + \Theta(n) = \Theta(n^2)$
    Worst than sorting!
Average Case:

- Assume **Partition**() Algorithm breaks $A$ to two pieces with sizes $k$ and $n-k-1$,

$$T(n) = \frac{1}{n} \sum_{k=0}^{n-1} T(\max(k, n-k-1)) + \Theta(n)$$

$$\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)$$

- Assume $T(n) \leq cn$ for some $c$.

---

Average Case (cont’d.)

$$T(n) = \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

$$= \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor} k \right) + \Theta(n)$$

$$= \frac{2c}{n} \left( \frac{n}{2} (n-1) - \frac{1}{2} \frac{n}{2} \left( \frac{n}{2} - 1 \right) \right) + \Theta(n)$$

$$= c(n-1) - \frac{c}{2} \left( \frac{n}{2} - 1 \right) + \Theta(n)$$

$$= cn - \left( \frac{cn}{4} + \frac{c}{2} - \Theta(n) \right)$$

$$\leq cn$$
Worst-case Linear-Time O.S.

Select($A, p, q, i$) Algorithm:
1. Divide $A$ to $n/5$ groups of size 5.
2. Find the median of each group of 5 by brute force, and store them in a set $A'$ of size $n/5$.
3. Use Select($A', 1, n/5, n/10$) to find the median $x$ of $n/5$ medians.
4. Partition the $n$ elements around $x$. Let $k = q-p+1$ (rank of $x$).
5. if $i=k$ then
   return $x$
   if $i<k$ then Select($A, p, q, i$)
   else Select($A, q, r, i-k$)

Analysis

- At least half of $A'$ is less than $x$, which is at least $n/10$ elements of $A'$.
- Thus $3n/10$ elements are smaller than $x$.
- If $n>50$ then $3n/10>n/4$, so $n/4$ elements are smaller than $x$, and we know where they are!
- The components of recurrence for $T(n)$:
  - $T(n/5)$: to find median of $n/5$ medians,
  - $T(3n/4)$: the complexity of step 5.
  - $\Theta(n)$: The time for Partition$()$.
  - $T(n) = T(n/5) + T(3n/4) + \Theta(n)$
Analysis (cont’d.)

- **Claim:** \( T(n) = cn. \)

\[
T(n) = \frac{cn}{5} + \frac{3cn}{4} + \Theta(n) \\
\leq 19cn/20 + O(n) \\
= cn - (cn/20 - O(n)) \\
\leq cn, \text{ for large enough } c.
\]

---

Simplified Master Theorem:

- Assume that \( T(1) = d \), and for \( n > 1 \):

\[
T(n) = aT(n/b) + cn.
\]
- If \( a < b \), Then \( T(n) = O(n) \);
- If \( a = b \), Then \( T(n) = O(n \log n) \);
- If \( a > b \), Then \( T(n) = O(n^{\log_b a}) \);

\[\text{e.g. } T(n) = 4T(n/2) + cn \text{ gives } T(n) = O(n^{\log_2 4}) = O(n^2)\]
Today’s Lecture

- Binary Search Trees
- Balanced Search Trees

The Structure

- Each node $x$ in a binary search tree (BST) contains:
  - $key[x]$ - The value stored at $x$.
  - $left[x]$ - Pointer to left child of $x$.
  - $right[x]$ - Pointer to right child of $x$.
  - $p[x]$ - Pointer to parent of $x$. 
**BST - Property**

- Keys in BST satisfy the following properties:
  - Let \( x \) be a node in a BST:
  - If \( y \) is in the left subtree of \( x \) then:
    \[ \text{key}[y] \leq \text{key}[x] \]
  - If \( y \) is in the right subtree of \( x \) then:
    \[ \text{key}[y] > \text{key}[x] \]

---

**Example:**

- Two valid BST’s for the keys: 2,3,5,5,7,8.
In-Order Tree walk

- Can print keys in BST with in-order tree walk.
- Key of each node printed between keys in left and those in right subtrees.
- Prints elements in monotonically increasing order.
- Running time?

In-Order Traversal

Inorder-Tree-Walk(x)
1: If \( x\neq NIL \) then
2: Inorder-Tree-Walk(left[x])
3: Print(key[x])
4: Inorder-Tree-Walk(right[x])

What is the recurrence for \( T(n) \)?
What is the running time?
In-Order Traversal

- In-Order traversal can be thought of as a projection of BST nodes on an interval.
- At most $2^d$ nodes at level $d=0, 1, 2, \ldots$

Other Tree Walks

Preorder-Tree-Walk$(x)$
1: If $x \neq \text{NIL}$ then
2: Print($key[x]$)
3: Preorder-Tree-Walk($left[x]$)
4: Preorder-Tree-Walk($right[x]$)

Postorder-Tree-Walk$(x)$
1: If $x \neq \text{NIL}$ then
2: Postorder-Tree-Walk($left[x]$)
3: Postorder-Tree-Walk($right[x]$)
4: Print($key[x]$)
Searching in BST:

- To find element with key \( k \) in tree \( T \):
  - Compare \( k \) with \( \text{root}[T] \)
  - If \( k < \text{key}[\text{root}[T]] \) search for \( k \) in \( T \)
  - Otherwise, search for \( k \) in \( \text{Search}(T,k) \)

\[
\begin{align*}
\text{Search}(T,k) & \\
1: & \text{ } x = \text{root}[T] \\
2: & \text{ If } x = \text{NIL} \text{ then return(”not found”) } \\
3: & \text{ If } k = \text{key}[x] \text{ then return(”found the key”) } \\
4: & \text{ If } k < \text{key}[x] \text{ then Search(left[x],k) } \\
5: & \text{ else Search(right[x],k) }
\end{align*}
\]

Examples:

- Search(\( T,11 \))
- Search(\( T,6 \))
Analysis of Search

- Running time of height $h$ is __________
- After insertion of $n$ keys, worst case running time of search is __________

BST Insertion

- Basic idea: similar to search.
- BST-Insert:
  - Take an element $z$ (whose right and left children are NIL) and insert it into $T$.
  - Find a place where $z$ belongs, using code similar to that of Search.
  - Add $z$ there.
Insert Key

BST-Insert($T, z$)
1: $y = \text{NIL}$
2: $x = \text{root}[T]$
3: While $x \neq \text{NIL}$ do
4:   $y = x$
5:   if key[$z$] < key[$x$] then
6:      $x = \text{left}[x]$
7:   else $x = \text{right}[x]$
8: $p[z] = y$
9: if $y = \text{NIL}$ then $\text{root}[T] = z$
10: else if key[$z$] < key[$y$] then \text{left}[y] = z
11: else \text{right}[y] = z

Locating the Minimum

BST-Minimum($T$)
1: $x = \text{root}[T]$
2: While $\text{left}[x] \neq \text{NIL}$ do
3:   $x = \text{left}[x]$
4: return $x$
Application: Sorting

- Can use BST-Insert and Inorder-Tree-Walk to sort list of \( n \) numbers

**BST-Sort**
1: root[\( T \)] = NIL
2: for \( i = 1 \) to \( n \) do
3: \( \text{BST-Insert}(T, A[i]) \)
4: \( \text{Inorder-Tree-Walk}(T) \)

| Sort Input: 5, 10, 3, 5, 7, 5, 4, 8 |
| Inorder Walk: 3, 4, 5, 5, 7, 8, 10 |

Analysis:

- The running time depends on the height of the tree (the Insert time).
- The average case analysis is like quick sort (which element will sit in the root).
- Therefore the expected running time is \( O(n \log n) \).
- Average BST height is \( O(\log n) \).
**Successor**

Given $x$, find node with smallest key greater than $key[x]$. Here are two cases depending on right subtree of $x$.

- **Successor Case 1:**
  - The right subtree of $x$ is not empty. Successor is leftmost node in right subtree. That is, we must return $BST\text{-Minimum}(right[x])$

```
BST-Successor(x)
1: If right[x]!=NIL then
2: return BST-Minimum(right[x])
3: y=p[x]
4: While (y!=NIL) and (x=right[y])
5: x=y
6: y=p[y]
7: return y
```

**Successor Case 2:** The right subtree of $x$ is empty. Successor is lowest ancestor of $x$. Observe that, “Successor” is defined as the element encountered by **inorder** traversal.

```
BST-Successor(x)
1: If right[x]!=NIL then
2: return BST-Minimum(right[x])
3: y=p[x]
4: While (y!=NIL) and (x=right[y])
5: x=y
6: y=p[y]
7: return y
Running time?
```
Deletion

- Delete a node $x$ from tree $T$:
  - Case 1: $x$ has no children.

```
  A
 /\  \\
B  D
  \  /
   C
```

Deletion:

- Case 2: $x$ has one child (call it $y$). Make $p[x]$ to replace $y$ instead of $x$ as its child, and make $p[x]$ to be $p[y]$.

```
  A
 /\  \\
B  D
  \  /
   C
```

```
  A
 /\  \\
B  D
  \
   C
```
Deletion:

- Case 3: \( x \) has two children:
  - Find its successor (or predecessor) \( y \).
  - Remove \( y \). (Note \( y \) has at most one child, why?)
  - Replace \( x \) by \( y \).

```
Delete Procedure

BSFT-Delete\( (T, z) \)
1: If \((\text{left}[z]=\text{NIL})\) or \((\text{right}[z]=\text{NIL})\) then
2: \( y = z \)
3: else \( y = \text{BST-Successor}(z) \)
4: If \( \text{left}[y] = \text{NIL} \) then
5: \( x = \text{left}[y] \)
6: else \( x = \text{right}[y] \)
7: If \( x = \text{NIL} \) then \( p[x] = p[y] \)
8: If \( p[y] = \text{NIL} \) then \( \text{root}[T] = x \)
9: else if \( y = \text{left}[p[y]] \) then \( \text{left}[p[y]] = x \)
10: else \( \text{right}[p[y]] = x \)
11: if \( y = \text{NIL} \) then \( \text{key}[z] = \text{key}[y] \)
12: return \( y \)
```