Lecture 3: Overview

- Read Chapters 1-3, 10
- Read Chapters 4, 6, 7, 8, 9
- HW: Problems 2-2, 3-1 a-c, and 4-1 a-d
- Recurrences with Summation
- Heap Sort
- Quick Sort
- Linear Time Sorting
- Median Finding
Recurrences with Summation

• Sometimes we derive bounds of the form:

\[ T(n) = \begin{cases} 
  \Theta(n) & \text{if } n = 1 \\
  2T(n/2) + \Theta(n) & \text{if } n > 1 
\end{cases} \]

• We’d like to figure out the solution in closed form
Substitution Method

• Guess the form of the solution
• Use mathematical induction to find the constants and show that the solution works
• Example:
  \[ T(n) = 2T(\lfloor n/2 \rfloor) + n \]
• Guess \( T(n) = O(n \cdot \log n) \)
• Prove \( T(n) \leq c \cdot n \cdot \log n \) for the appropriate choice of \( c \)
Substitution Method

\[ T(n) \leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \]
\[ \leq cn \lg(n/2) + n \]
\[ = cn \lg n - cn \lg 2 + n \]
\[ = cn \lg n - cn + n \]
\[ \leq cn \lg n \]

- **Base case**
  \[ T(1) = c \lg 1 = 0 \]

Take advantage that asymptotic notation only requires us to prove \( T(n) \leq cn \lg n \) for \( n > n_0 \)

Instead use the base case \( T(2) \leq c \ 2 \lg 2 \)
Recursion Trees

- Consider $T(n) = 3T(n/4) + cn^2$

Tree representation:

```
     cn^2
   /       \       /
T(n/4)   T(n/4)   T(n/4)
```
Recursion Trees

- Consider \( T(n) = 3T(n/4) + cn^2 \)

\[ T(n) \]

\[ \begin{array}{c}
\text{cn}^2 \\
\text{c(n/4)}^2 \quad \text{c(n/4)}^2 \quad \text{c(n/4)}^2 \\
T(n/16) \\ T(n/16) \\ T(n/16) \\ T(n/16) \\ T(n/16) \\ T(n/16) \\ T(n/16) \\ T(n/16) \end{array} \]
Recursion Trees

Consider $T(n)=3T(n/4)+cn^2$

$T(n)$

$\log_4 n$

$cn^2$

$c(n/4)^2$

$c(n/4)^2$

$c(n/4)^2$

$3cn^2/16$

$3^2cn^2/16^2$

$\Theta(n^{\log_4 3})$

$n^{\log_4 3}$
Recursion Trees

\[ T(n) = cn^2 + \frac{3}{16} cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \]

\[ = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \]

\[ = \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \]

\[ = O(n^2) \]
Master Method

- Method to solve recurrences of the form \( T(n) = aT(n/b) + f(n) \) where \( a \geq 1 \) and \( b > 1 \) are constants and \( f(n) \) is an asymptotically positive function.

- Recurrence describes the runtime of an algorithm that divides a problem of size \( n \) into \( a \) subproblems, each of size \( n/b \), where \( a \) and \( b \) are positive constants.
Master Theorem

Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function, and let \( T(n) \) be defined on the nonnegative integers by the recurrence

\[
T(n) = aT(n/b) + f(n),
\]

where we interpret \( n/b \) to mean either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \).

Then \( T(n) \) can be bounded asymptotically as follows.

1. If \( f(n) = O(n^{\log a/\log b - \varepsilon}) \) for some constant \( \varepsilon > 0 \), then
   \[
   T(n) = \Theta(n^{\log a/\log b}).
   \]

2. If \( f(n) = \Theta(n^{\log a/\log b}) \), then
   \[
   T(n) = \Theta(n^{\log a/\log b \log n}).
   \]

3. If \( f(n) = \Omega(n^{\log a/\log b + \varepsilon}) \) for some constant \( \varepsilon > 0 \), and if
   \[
   a f(n/b) \leq cf(n)
   \]
   for some constant \( c < 1 \) and all sufficiently large \( n \), then
   \[
   T(n) = \Theta(f(n)).
   \]
Heaps

• Binary Tree w/ keys stored at its nodes
• Two properties:
  • Heap-order: for every node \( v \) other than the root
    \[ \text{key}(v) \geq \text{key}(\text{parent}(v)) \]
  • Complete Binary Tree: for a tree of height \( h \)
    • for \( i = 0 \) to \( h-1 \) there are \( 2^i \) nodes of depth \( i \)
    • at depth \( h \), the nodes are to the left
Heaps

```
     1
    / \    \
   3   4
   / \   \  
  6   5
```
Heaps

- A heap with $n$ nodes has height $O(\log n)$
- Depth $i$ has $2^i$ nodes
- $1+2+4+8...+2^{h-2}+1$ (at least one at $h-1$) = $2^{h-1}$ nodes
- $n \geq 2^{h-1}$
- $\log(n) \geq h-1$
- $h$ is $O(\log(n))$
Insertions

1. Put new value at last node of heap
2. Swap new value and parent if new value is greater than parent
3. Repeat 2 until either the root is reached or no swap is performed
Removal

- Root has the smallest value
  1. Remove root and move last node’s value to the root
  2. If the root node is smaller than either of its children swap it with the smaller child
  3. Repeat 2 until either no swap is needed or until we reach a leaf node
Array based Heap

- Can store the heap in an array
- For node $i$
  - Left child is stored at location $2^i$
  - Right child is stored at location $2^i+1$
Merging Heaps

- Given two heaps and a key k
- Create new root with key k and children roots of the other two heaps
- Push k downwards to restore heap order property
Bottom Up Heap Construction

- Can use this merge procedure to construct a heap
- Start with many heaps of size 1
- Merge them in stages all to make one heap
- Stage 1 constructs heaps of size 3
- Stage 2 constructs heaps of size 7
- ...
Cost of Bottom Up Construction

- Each root node in the first stage can be pushed down 1 level
- Each root node in the second stage can be pushed down 2 levels
- ...
- Total cost is $O(n)$
- Can graphically see this by imagining that each root value goes right first then left -> each node is traversed by two paths -> cost is $O(n)$
Heapsort

• Can do heap sort in place
• Book gives details
• Basic idea:
  • Construct heap in place in array
  • Remove item from heap and place it at the end of the array
  • Repeat until the entire array is sorted
Priority Queue

- Stores a collection of items
- Each item contains both an element and key
- Main methods:
  - insertItem(k, o)
  - removeMin()
- Additional Methods:
  - minKey()
  - minElement()
  - size(), isEmpty()
- Applications:
  - Stock market (buy a stock at the cheapest price)
  - Standby flyers (platinum fliers get priority)
Comparing Keys

• We want to abstract the concept of comparing
• Provide a comparator for comparing keys
  • Contains methods that return whether keys are less than, equal, or greater than
  • Contains method that return whether comparator can compare a given key
Sorting with Priority Queues

- Can use priority queues to sort by key
- Algorithm:
  1. For a key in the input set
  2. Take a key and insert it into queue
  3. Until queue isEmpty()
  4. Remove the minimum key
Quicksort

- Idea: pick an element in list
- Split list into two groups: one smaller than the element, one greater
- Recursively use quick sort on each group
Worst Case Performance

- Uneven splitting
- All elements are either greater or less than the pivot point
- Could have \( n-1 \) rounds
- \( T(n) = T(n-1) + T(0) + \Theta(n) \)

\[
\sum_{k=1}^{n} k = \frac{1}{2} n(n+1) = \Theta(n^2)
\]
Best Case

- Even partitioning
- \( T(n) \leq 2T(n/2) + \Theta(n) \)
- Use 2nd case of Master theorem:
  If \( f(n) = \Theta(n^{\log a / \log b}) \), then \( T(n) = \Theta(n^{\log a / \log b} \lg n) \).
- \( T(n) = \Theta(n^{\log 2 / \log 2} \lg n) = \Theta(n \ lg n) \)
Lower Limit on Worst Case Time Bound on Sorting

- It turns out that we can show a lower limit on the worst case time bound on sorting for comparison sorts.
- Without loss of generality, we can consider that all of the input elements are distinct. This makes all comparison operations equivalent.
Decision Tree

1:2

2:3
  ≤
  1:3
  ≤
  <1,3,2>

1:3
  >
  <2,1,3>

2:3
  >
  ≤
  <3,2,1>

1:3
  ≤
  <2,3,1>

≤

>
Worst Case Bounds

- For $n$ entries, there are $n!$ permutations
- Consider a decision tree with height $h$ and $l$ reachable leaves
- A tree of height $h$ has no more than $2^h$ nodes
- Therefore: $n! \leq l \leq 2^h$
- $h \geq \lg(n!)$
- $h = \Omega(n \lg n)$ (use Stirling’s approximation)
Counting Sort

• Assumes the input elements are integers in the range 0 to k
• If $k = O(n)$, the sort is $O(n)$
• Idea: Compute for each integer $i$ between 0 through $k$ the number of occurrences of integers less than or equal to $i$ ($O(n)$)
• Use this array to place elements in the input into sorted order $O(n)$
Radix Sort

• Works on integers with $d$-digits

• Idea:
  • Sort (bin) integers into bins based upon least significant digit
  • Resort based upon next least significant
  • Continue until most significant digit

• $O(n)$ sort
Radix Sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>231</td>
<td>534</td>
<td>223</td>
</tr>
<tr>
<td>979</td>
<td>311</td>
<td>421</td>
<td></td>
</tr>
</tbody>
</table>
Radix Sort

231
311
421
123
223
534
979
<table>
<thead>
<tr>
<th>Radix Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>311</td>
</tr>
<tr>
<td>421</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>223</td>
</tr>
<tr>
<td>231</td>
</tr>
<tr>
<td>231</td>
</tr>
<tr>
<td>534</td>
</tr>
<tr>
<td>979</td>
</tr>
</tbody>
</table>
Radix Sort

123
223
231
311
421
534
979
Bucket Sort

• Idea: assume values to be distributed over some range, split range up into n equal sized buckets
• Example:
  • assume 10 values are distributed over [0..1].
  • We split values in to 10 buckets: [0..0.1], [0.1..0.2], [0.2..0.3]...[0.9..1]
• Go through input and separate into buckets
• Sort each bucket using insertion sort
Cost Analysis

\[ T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \]

\[ E[T(n)] = E \left[ \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \right] \]

\[ = \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \quad \text{(linearity)} \]

\[ = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \]
Cost Analysis

- Define indicator random variable $X_{ij} = I\{A[j] \text{ falls in bucket } i\}$ for $i=0,1,...,n-1$ and $j=1,2,...,n$
Cost Analysis

\[ n_i = \sum_{j=1}^{n} X_{ij} \]

\[ E[n_i^2] = E \left[ \left( \sum_{j=1}^{n} X_{ij} \right)^2 \right] \]

\[ = E \left[ \sum_{j=1}^{n} \sum_{k=1}^{n} X_{ij}X_{ik} \right] \]

\[ = E \left[ \sum_{j=1}^{n} X_{ij}^2 + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n \atop k \neq j} X_{ij}X_{ik} \right] \]

\[ = \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n \atop k \neq j} E[X_{ij}X_{ik}] \]
Cost Analysis

\[ E[X_{ij}^2] = 1(1/n) + 0(1-1/n) \]
\[ = 1/n \]
\[ E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}] \]
\[ = (1/n)(1/n) \]
\[ = 1/n^2 \]
Cost Analysis

\[
\sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n, k \neq j} E[X_{ij}X_{ik}]
\]

\[
= n \frac{1}{n} + n(n - 1) \frac{1}{n^2}
\]

\[
= 1 + \frac{n - 1}{n}
\]

\[
= 2 - \frac{1}{n}
\]
Cost Analysis

• Therefore bucket search is $O(n)$
Min/Max/Both

- Find min or max requires n-1 comparison
  1. Pick first value as min or max
  2. Compare current min or max to all values in the array
  3. Update current min or max if we find a smaller or larger value

n-1 comparisons
How many for both min/max?

- Trivial algorithm takes $2n-2$ (run both min and max)
- Can do in $\lceil 3n/2 \rceil$
- Idea: compare pairs of integers
- If $n=\text{odd}$, set the initial min/max to the first integer, then compare the remaining integers pairwise. Then search for min in the smaller value set and max in the larger value set.
- If $n=\text{even}$, compare pairwise, then run the min algorithm on the smaller value set and the max on the larger value set.
Selecting the ith Element

• We want to select the ith smallest element
• Can use quicksort type algorithm (with randomized node selection)
• Basic idea:
  1. Select random pivot point
  2. Split nodes around pivot point
  3. Run procedure on set of nodes that contains ith smallest element
Complexity Analysis

- Best Case: $O(n)$ - pivot point is the $i$th element
- Even partition:
  $T(n) = O(n) + O(n/2) + O(n/4) + \ldots + O(2) + O(1) = O(2n) = O(n)$
- Worst case:
  $T(n) = O(n) + O(n-1) + \ldots + O(2) + O(1) = O(n^2)$
Complexity Analysis

- **Average time:**
  1/n probability of each position i as pivot.

\[
T(n) = \sum_{i=1}^{n-1} \frac{1}{n} T(\max(i, n-i)) + O(n)
\]

\[
= n \sum_{i=[n/2]}^{n-1} \frac{2}{n} T(i) + O(n)
\]

Guess \( T(n) \leq cn \)

\[
T(n) \leq \frac{2}{n} \sum_{i=[n/2]}^{n-1} ci + an = \frac{2c}{n} \left( \sum_{k=1}^{n} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an
\]

\[
= \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) + an \leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(n/2 - 2)(n/2 - 1)}{2} \right)
\]

\[
= c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \leq cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right)
\]
Complexity Analysis

• Need to show that for large $n$, last expression is at most $cn$. To do this, $cn/4-c/2-an \geq 0$.
• Choose $c > 4a$
• Gives $n \geq 2c/(c-4a)$
Select w/ worst-case linear time

- Idea: split input into sets of 5
- Compute median of each of these sets
- Use select to compute the median of these medians
- Pivot around this point
Select w/ linear worst-case

- Pivot point is guaranteed to be good:
- Note that half of medians are less than or equal to the chosen median of medians and half are greater or equal
- Each of these sets contributes either 3 elements that are less than or equal to the median of medians or 3 elements that are greater than or equal to the median of medians
- Book contains proof that this algorithm is $O(n)$