CSE 373

APRIL 3RD – ALGORITHM ANALYSIS

ASSORTED MINUTIAE

- Drawn notes from last week
- HW1P1 due Wed at midnight
- HW1P2 due Fri at midnight
- Additional Java review?

TODAY'S SCHEDULE

- Finish discussion of heaps
- Algorithm analysis
- Analyzing the heap

REVIEW FROM LAST WEEK

- **Priority queue**
  - Data inserted with priority
  - Lower items dequeue first
  - Can change priorities of items
  - FIFO is sacrificed in implementation

REVIEW FROM LAST WEEK

- **Heap**
  - Tree structure with two properties:
    - Completeness: Filled from left to right, top to bottom
    - Heap property: Parents are smaller than their children (min-heap)

REVIEW FROM LAST WEEK

- **Percolate up**
  - When a new item is inserted:
    - Place the item at the next position to preserve completeness
    - Swap the item up the tree until it is larger than its parent

REVIEW FROM LAST WEEK

- **Percolate down**
  - When an item is deleted:
    - Remove the root of the tree (to be returned)
    - Move the last object in the tree to the root
    - Swap the moved piece down while it is larger than it's smallest child
    - Only swap with the smallest child

HEAPS AS ARRAYS

- **Because heaps are complete, they can be represented as arrays without any gaps in them.**
- **Naïve implementation:**
    - Left child: 2*i+1
    - Right child: 2*i + 2
    - Parent: (i-1)/2

HEAPS AS ARRAYS

- **Alternate (common) implementation:**
  - Put the root of the array at index 1
  - Leave index 0 blank
  - Calculating children/parent becomes:
    - Left child: 2*i
    - Right child: 2*i + 1
    - Parent: i/2

HEAPS AS ARRAYS

- **Why do an array at all?**
  - + Memory efficiency
  - + Fast accesses to data
  - + Forces log n depth
  - - Needs to resize
  - - Can waste space
- **Overall, however, better done through an array**

ALGORITHM ANALYSIS

- **Important topic. Why?**
  - Show that an implementation is better.
- **What do we mean by better?**
  - Fewer clock cycles
  - More efficient memory usage
  - Correctness

ALGORITHM ANALYSIS

- **Math review**
- **Logarithms**
  - $\log_2 x = y$ when $x = 2^y$
  - How does this grow? Slowly
  - A balanced tree has a height ~$\log_2 n$
  - $\log_k x$ differs from $\log_j x$ by a constant factor

ALGORITHM ANALYSIS

- **Floor and ceiling**
  - Integer rounding, computers operate in integer quantities
    - Clock cycles
    - Memory bytes

ALGORITHM ANALYSIS

- **Floor and ceiling**
  - Integer rounding, computers operate in integer quantities
    - Clock cycles
    - Memory bytes

Floor : $\lfloor X \rfloor$ **denotes largest integer $\leq$ x**

Ceiling: $\lceil X \rceil$ **denotes smallest integer $\geq$ x**

ALGORITHM ANALYSIS

- **Operations**
  - Arithmetic
  - Comparisons
  - Memory reads/writes
- **Loops and functions are just chains of these operations.**

ALGORITHM ANALYSIS

```
Int value = 0;
for(int i = 0; i < 10; i++){
     value++;
}
```

How long does this take?

ALGORITHM ANALYSIS

```
Int value = 0;
for(iint i = 0; i < N; i++){
      value++;
}
```

How long does this take?

ALGORITHM ANALYSIS

- **Principles of analysis**
  - Determining performance behavior
  - How does an algorithm react to new data or changes?
  - Independent of language or implementation

ALGORITHM ANALYSIS

- Example: find()
- Suppose an array with 5 elements
- One implementation has a sorted array, the other is unsorted
- For which one will find() be faster?
- How long will it take?

ALGORITHM ANALYSIS

- **Find(1)**

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- Find(1)
- How many operations?

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- **Find(4)?**

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- Not a good representation of how the algorithm actually behaves.
- Want to access the algorithm on the whole, not just over a few inputs
- This is why testing alone isn't enough

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: find the average performance over all inputs
  - Worst case: how long the program takes to complete the worst case problems.

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: can be difficult to compute

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: can be difficult to compute
  - What is the average case for binary search?

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Worst case: is most commonly used
  - Easily compared and gives a good estimate of the robustness of an algorithm

ALGORITHM ANALYSIS

- **Worst case runtime here?**

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- **Worst case runtime here?**
- **Are we convinced one is better just looking at 5 elements?**

1	2	3	4	5			

4	2	5	3	1			

ASYMPTOTIC ANALYSIS

- Want to know how algorithms behave with big data
- How much more does an additional element in our data structure cost us?

ASYMPTOTIC ANALYSIS

- **Consider find() for sorted v. unsorted arrays**
  - Which is better?
  - Unsorted grows linearly – if we add one more element to the list, we expect that the algorithm will take one more operation to complete
  - How much longer is an extra element in the sorted case?

ASYMPTOTIC ANALYSIS

- **Consider find() for sorted v. unsorted arrays**
  - As trees grow exponentially in size
    they grow logarithmically in height
  - Height is what determines our runtime

ASYMPTOTIC ANALYSIS

- **Consider find() for sorted v. unsorted arrays**
  - We call the unsorted case: linear time or O(n) time
  - We call the sorted case: logarithmic time or O(log n) time

NEXT CLASS

- Formalizing big-O notation
- Looking at heaps and other algorithms