**CS330 – Sorting and Lower Bounds**

**Purpose:** We will examine algorithms that sort an array of values. There are many, many sorting algorithms so it is important to understand how well they work and the advantages and disadvantages of various algorithms. We also want to have a lower bound for sorting.

**Knowledge:** This activity will help you become familiar with the following content knowledge:

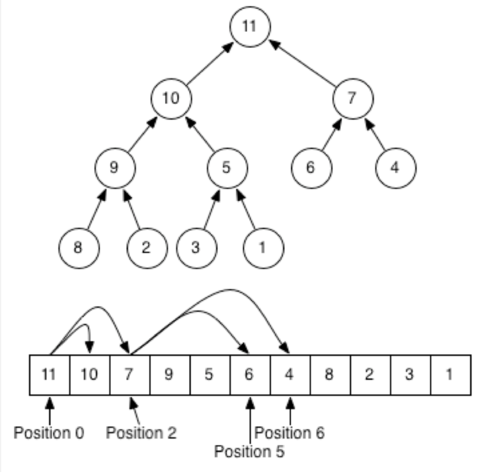
- Swap sorting algorithms
- The heap sort algorithm
- The quicksort algorithm
- Counting sort algorithms
- The lower bound for comparison sort algorithms

**Activity:** With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. Swap sorts of various kinds, like bubble sort and selection sort, repeatedly scan input, swapping any out-of-order elements. Inversions of an element are the number of smaller elements to the right of the element. The sum of inversions for all elements is the number of swaps required by bubble sort. Any algorithm that removes one inversion per swap requires at least this many swaps.

   What is the worst number of total inversions? What input causes this to happen?

2. A heap is a complete binary tree with the value at any node being at least as large as its two children.



Since a heap is a complete binary tree, it can be stored in an array, as we see above.

How would we compute the functions `leftChild(i)`, `rightChild(i)`, and `parent(i)` which give the appropriate index for the new element related to the one at position $i$?

3. Here is a sorting algorithm, heap sort, which uses this heap property.

Algorithm:

```
Build the heap
Repeat $n$ times:
    Remove the root
    Repair the heap
```

To build the heap you must insert $n$ nodes into the initially empty heap. What is the order of growth for building a heap? What is the cost for repairing a heap? What is the overall order of growth cost for the heap sort algorithm?
Hint: We saw this material in CS119.

4. Here is a sorting algorithm, quicksort

   Algorithm:

      Pick a pivot value

      Split the array into elements less than the pivot and elements greater than the pivot

      Recursively sort the two sublists

   Here is a beautiful implementation of this algorithm in Haskell:

```
quicksort [] = []
quicksort (x:xs) = quicksort small ++ (x: quicksort large) where
                        small = [y | y <- xs, y<= x]
                        large = [y | y <- xs, y>x]
```

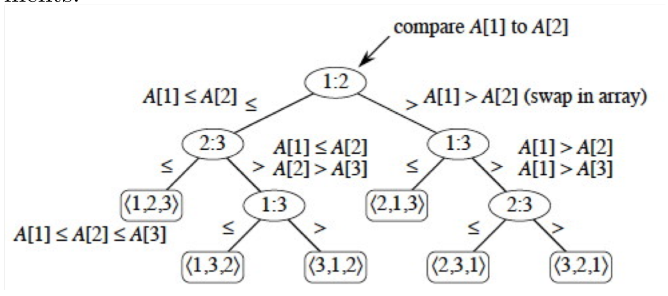
   What is the cost of doing the split with the pivot?
   What is the worst case for picking the pivot?
   So what is the worst case for quicksort?

5. Quicksort doesn't look that great on the worst case but we can determine the average case by looking at the recurrence relation:

$$f(n) = \begin{cases} 0 & n \leq 1 \\ n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} f(i) & n > 1 \end{cases}$$

   With a LOT of math crunching the closed form for $f(n)$ is $O(n \log n)$. You can read about this if you desire.

6. We have now seen a couple of $O(n \log n)$ sorting algorithms. To see if this is optimal we need a lower bound proof. Look at a possible decision tree for sorting three elements:



Why must there be $n!$ leaves in a sorting decision tree if we have $n$ elements?
Any binary tree of height $h$ has $k \leq 2^h$ leaves. Prove this by induction on $h$.
So a binary tree with $n!$ leaves must have a height of at least what?
It turns out that $\log n!$ is $O(n \log n)$ so why does this show that this is the lower bound for sorting?

7. If we know something about the structure of the data, it is possible to sort it without comparing elements to each other.

   Counting sort requires that the keys being sorted are in a known range $\{0, 1, ..., k\}$:

   > For each element in the input determine how many elements are less than it by performing a count.
   >
   > Place each of the elements directly in the position in an array determined by the count

   What is the order of growth and why does this not violate the lower bound that we have already proved?

**Activity:** With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

Read about properties of the following sorting algorithms in the many online resources and see what properties you can discover about the following sorting algorithms:

- insertion sort
- selection sort
- bubble sort
- heap sort
- quick sort
- merge sort
- radix sort

You don't need to focus on how the sorts work but instead concentrate on input cases that cause the sort to work poorly and/or well. Consider the amount of time and space that is required on average, on worst case, and on certain special cases. Are the sorts stable? You can look up what it means for a sort to be stable.

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies.

**Assignment 1**:
Read about the properties of the following sorting algorithms in the many online resources that you can find and then select the sort algorithm(s) that you would use in the following scenarios. Clearly explain what property of the sorting algorithm(s) makes it a good choice.

- insertion sort
- selection sort
- bubble sort
- heap sort
- quick sort
- merge sort
- radix sort

1. You are working on an embedded device (an ATM) that only has 4KB (4,096 bytes) of free memory, and you wish to sort the 2,000,000 transactions withdrawal history by the amount of money withdrawn (discarding the original order of the transactions
2. You are running a library catalog. You know that the books in your collection are almost in sorted ascending order by title, with the exception of one book which is in the wrong place. You want the catalog to be completely sorted in ascending order
3. To determine which of your Facebook friends were early adopters, you decide to sort them by their Facebook account ids, which are 64-bit integers. (Recall that you are super popular, so you have very many Facebook friends.)
4. You have a couple of thousand records of information that contain several fields including name, address, and grades. The records are already sorted by name. You want to sort the information by grade but want to make sure that records with the same grade will be listed in order of the sorted names.

**Criteria for Success:** You have a clear explanation of what sort you prefer for each of the scenarios, giving the properties of the sort that make it a good choice. There may be more than one sort that would be good so you need to justify your choice.