1. (10 pts) Suppose that we modify the `Partition` algorithm in QuickSort in such a way that on on every third level of the recursion tree it chooses the worst possible pivot, and on all other levels of the recursion tree `Partition` chooses the best possible pivot. Write down a recurrence relation for this version of QuickSort and give its asymptotic solution. Then, give a verbal explanation of how this `Partition` algorithm changes the running time of QuickSort.

2. (10 pts) Mr. Ollivander, of Ollivanders wand shop, has hired you as his assistant, to find the most powerful wand in the store. You are given a magical scale which "weighs" wands by how powerful they are (the scale dips lower for the wand which is more powerful). You are given n wands $W_1, \ldots, W_n$, each having distinct levels of power (no two are exactly equal).

   (a) Consider the following algorithm to find the most powerful wand:

      i. Divide the n wands into $\frac{n}{2}$ pairs of wands.
      ii. Compare each wand with its pair, and retain the more powerful of the two.
      iii. Repeat this process until just one wand remains.

   Illustrate the comparisons that the algorithm will do for the following $n = 8$ input:

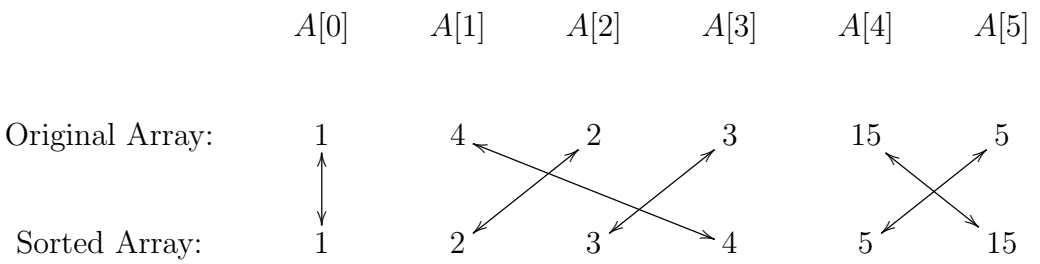
   $$W_1 : \tfrac{3}{2}, \quad W_2 : \tfrac{5}{2}, \quad W_3 : \tfrac{1}{2}, \quad W_4 : 1, \quad W_5 : 2, \quad W_6 : \tfrac{5}{4}, \quad W_7 : \tfrac{1}{4}, \quad W_8 : \tfrac{9}{4}$$

   (b) Show that for n wands, the algorithm (2a) uses at most n comparisons.

   (c) Describe an algorithm that uses the results of (2a) to find the *second* most powerful wand, using at most $\log_2 n$ additional comparisons. There is no need for pseudocode; just write out the steps of the algorithm like we have written in (2a). Hint: if you follow sports, especially wrestling, read about the *repechage*.

   (d) Show the additional comparisons that your algorithm in (2c) will perform for the input given in (2a).

3. (20 pts) For obtuse historical reasons, Prof. Dumbledore asks his students to line up in ascending order by height in a very tight room with little extra space. Similar to Alex the African Grey parrot (look it up!), the students, being bored, decided to play a little trick on Prof. Dumbledore. They lined up in order by height—*almost*. They made sure that each person was no more than k positions away from where they were supposed to be (in ascending order), but this allowed them to significantly mess up the precise ordering. Here is an example of an array with this property when $k = 2$:



(a) Write down pseudocode for an algorithm that would sort such an array in place—so it fits in the tight room—in time $O(n\,k \log k)$. Your algorithm can use a function `sort`(A, ℓ, r) that sorts the subarray $A[\ell], \ldots, A[r]$ in place in $O((r-\ell)\log(r-\ell))$ steps (assuming $r > \ell$).

(b) Suppose you are given to an auxiliary room which can fit $k+1$ students. Modify your previous algorithm to sort the given array in time $O(nk)$.

(c) With the same extra room as in the previous part, modify heap sort using a binary min heap of size $k+1$ to sort the given array in time $O(n \log k)$.

(d) (5 pts extra credit) Include the correct story about Alex, with proper citation. If you wish, you may copy this story verbatim, but must indicate clearly that you have done so and, of course, still cite your source.

4. (20 pts) Consider the following strategy for choosing a pivot element for the `Partition` subroutine of QuickSort, applied to an array A.

- Let n be the number of elements of the array A.
- If $n \leq 24$, perform an Insertion Sort of A and return.
- Otherwise:
  - Choose $2\lfloor n^{(1/3)} \rfloor$ elements at random from n; let S be the new list with the chosen elements.
  - Sort the list S using Insertion Sort and use the median m of S as a pivot element.
  - Partition using m as a pivot.
  - Carry out QuickSort recursively on the two parts.

(a) How much time does it take to sort S and find its median? Give a Θ bound.

(b) If the element m obtained as the median of S is used as the pivot, what can we say about the sizes of the two partitions of the array A?

(c) Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.

5. (20 pts extra credit) Recall that the *Insertion Sort* algorithm (Chapter 2.1 of CLRS) is an in-place sorting algorithm that takes $\Theta(n^2)$ time and $\Theta(n)$ space. In this problem, you will learn how to *instrument* your code and how to perform a numerical experiment that verifies the asymptotic analysis of Insertion Sort's running time. There are two functions and one experiment to do this.

   (i) `InsertionSort(A,n)` takes as input an unordered array A, of length n, and returns both an in-place sorted version of A and a count t of the number of atomic operations performed by `InsertionSort`.

   Recall: atomic operations include mathematical operations like $-$, $+$, $*$, and $/$, assignment operations like $\leftarrow$ and $=$, comparison operations like $<$, $>$, and $==$, and `RAM` indexing or referencing operations like $[\,]$.

   (ii) `randomArray(n)` takes as input an integer n and returns an array A such that for each $0 \leq i < n$, $A[i]$ is a uniformly random integer between 1 and n. (It is okay if A is a random permutation of the first n positive integers; see the end of Chapter 5.3.)

   (a) From scratch, implement the functions `InsertionSort` and `randomArray`. You may not use any library functions that make their implementation trivial. You may use a library function that implements a pseudorandom number generator in order to implement `randomArray`.

   Submit a paragraph that explains how you instrumented `InsertionSort`, i.e., explain which operations you counted and why these are the correct ones to count.

   Hint: your instrument code should only count the operations of the `InsertionSort` algorithm and not the operations of the instrument code you added to it.

   (b) For each of $n = \{2^3, 2^5, \ldots, 2^{15}, 2^{16}\}$, run `InsertionSort(randomArray(n),n)` fives times and record the tuple $(n, \langle t \rangle)$, where $\langle t \rangle$ is the average number of operations your function counted over the five repetitions. Use whatever software you like to make a line plot of these 12 data points; overlay on your data a function of the form $T(n) = A\,n^2$, where you choose the constant A so that the function is close to your data.

   Hint: To increase the aesthetics, use a log-log plot.