1. Design 3 algorithms based on binary min heaps that find the $k$th smallest # out of a set of $n$ #’s in time:

   a) $O(n \log k)$
   b) $O(n + k \log n)$
   c) $O(n + k \log k)$

Use the heap operations (here $s$ is the size):

- Insert, delete: $O(\log s)$
- Buildheap: $O(s)$
- Smallest: $O(1)$

Give high level descriptions of the 3 algorithms and briefly reason correctness and running time. Part c) is the most challenging.

2. Consider the following sorting algorithm for an array of numbers (Assume the size $n$ of the array is divisible by 3):

   - Sort the initial 2/3 of the array.
   - Sort the final 2/3 and then again the initial 2/3.

Reason that this algorithm properly sorts the array. What is its running time?

3. KT, problem 1, p 246.

4. Suppose you are choosing between the following 3 algorithms:

   (a) Algorithm A solves problems by dividing them into 5 subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.

   (b) Algorithm B solves problems of size $n$ by recursively solving 2 subproblems of size $n - 1$ and the combining the solutions in constant time.

   (c) Algorithm C solves problems of size $n$ by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and the combining the solution in $O(n^2)$ time.

What are the running times of each of these algs. (in big-O notation), and which would you choose?

5. (a) Compute the FFT of the polynomial $1 + 2x - x^3$ by computing the 4 dimensional FFT matrix and multiplying it with the coefficient vector $[1 \ 2 \ 0 \ -1]^\top$.

   The FFT matrix uses powers of a root of unity. First determine the appropriate root of unity.
(b) Now compute the inverse FFT of the vector $[1 \ 2 \ 0 \ -1]^\top$. Again find the appropriate matrix and multiply this matrix by the vector.

(c) Check that the two matrices used above are inverses of each other.

6. (Extra Credit) The square of a matrix $A$ is its product with itself, $AA$.

(a) Show that 5 multiplications are sufficient to compute the square of a $2 \times 2$ matrix.

(b) What is wrong with the following algorithm for computing the square of an $n \times n$ matrix. “Use a divide-and-conquer approach as in Strassen’s algorithm, except that instead of getting 7 subproblems of size $n/2$, we now get 5 subproblems of size $n/2$ thanks to part a). Using the same analysis as in Strassen’s algorithm we can conclude that the algorithm runs in time $O(n^{\log_2 5})$.”

(c) In fact, squaring matrices is no easier than matrix multiplication. Show that if $n \times n$ matrices can be squared in time $O(n^c)$, then any two $n \times n$ matrices can be multiplied in time $O(n^c)$. 