Due date: Tuesday, April 6, 9:59 AM. You will need to submit this via GradeScope. Instructions for this are in the course syllabus.

For this assignment, each answer must be contained within a single piece of paper, although you are allowed to use multiple pages for the assignment. When you submit to GradeScope, you will need to inform the system which page of your scanned PDF contains the answer. *Do this even if your submission is a single page. Failure to do so may cost you points.*

Please review the syllabus and course reference for the expectations of assignments in this class. Remember that problem sets are not online treasure hunts. You are welcome to discuss matters with classmates, but remember the Kenny Loggins rule. Remember that you may not seek help from any source where not all respondents are subject to UC Irvine's academic honesty policy.

1. Recall the linear search algorithm:

```
int linearSearch(const std::vector<int> & numbers, int target)
{
    int i;
    int n = numbers.size();
    for(i=0; i < n; i++)
    {
        if( numbers[i] == target )
        {
            return i;
        }
    }
    throw ElementNotFoundException("Element not found by linear search.");
}
```

   Would it be correct to describe this algorithm as having a running time of $\Omega(n)$, where n is the size of the input vector? In 1-3 sentences, justify your answer.

2. Recall the following algorithm, which determines if a given input positive integer is prime:

```
bool isPrime?(positive integer n)
  if n = 1 then
    return false
  else if n is 2 then
    return true
  for i from 3 to ⌈√n⌉ by twos do
    if i divides n leaving no remainder then
      return false
  return true
```

   (a) Express the running time of this algorithm in $\mathcal{O}$ notation in terms of n, the value of the input number provided. Assume that each call to the divisibility test inside the for loop takes constant time.

   (b) Is this a polynomial time algorithm, according to the definition of that term given in lecture? In 1-3 sentences, justify your answer.

3. We say a positive integer n is "resolute" if 3 evenly divides (that is, leaves no remainder when used as a divisor) $n^2 + 2n$. I put forth a claim that all odd positive integers are resolute. Demonstrate that this claim is incorrect.

4. Suppose we have an array A of n professors; each teacher has two characteristics: difficulty (a real number in the range $[0, 10]$, where a higher number indicates a more difficult teacher) and humor (a real number in the range $[0, 100]$, where a higher number indicates a funnier teacher). You may assume that each value is distinct (no two professors are exactly as difficult or exactly as funny), but not that the precision of real numbers is limited (as floats and doubles are in C++ and Java). Our goal is to determine a set of professors that might satisfy an answer to the question "who is the easiest teacher that is the funniest?" We wish to avoid professors with high difficulty and low humor. We would like to find the largest subset of the input data such that no professor in the chosen subset is **both** less funny **and** more difficult than another professor in the original input set. Note that if professor A is easier than professor B, it *does not follow* that professor A is also funnier than professor B.

   For example, if our dataset is:

Professor	Difficulty	Humor
Venabili	4	65
Jones Jr	8	15
Jones Sr	8.5	2
Grant	2	35
Moriarty	10	0
Plum	9	85
Walsh	8.2	90

   Then we want to return Venabili, Grant, and Walsh.

   Note that none of these are meant to refer to anyone at UC Irvine and any similarity is unintentional and a coincidence.

   (a) Devise an algorithm which takes as input A and n, and outputs the resulting set. Your algorithm does not need to be particularly efficient, but you may not give an algorithm that enumerates every subset.

   Do not have your answer depend on limitations of any programming language. For example, while difficulty or humor are numeric types, do not use that, say, a **double** in C++ is "not really" a real number.

   (b) Analyze the worst-case runtime of your algorithm using Θ-notation.

   (c) Do you believe your algorithm has obtained the best possible asymptotic runtime? Explain your reasoning. Note that finding the best possible asymptotic runtime **is not** required to get full credit on this question.

5. Suppose you have two max-heaps, A and B, with a total of n elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time $\mathcal{O}(n \log n)$. For this problem, do not use the fact that heaps are arrays. Rather, use the API of a heap; that is, you may call the following functions:

   - `max()`, which returns the maximum element in the heap at the moment.
   - `extractMax()`, which removes the maximum element in the heap.
   - `insert(e)`, which inserts the parameter into the heap.

   Each of these functions takes the time that they did when you learned them in a course like ICS 46. The heaps are implemented as binary heaps via an array (or equivalent).

   Give a brief explanation for why your algorithm has the required running time.

Note that, while this homework covers many core concepts for unit one of the class, it is not a comprehensive evaluation of your knowledge of the material thus far. You should still review lectures and the algorithms and concepts covered within for the first exam. For example, despite a lack of a question on this homework about Selection Sort, you should still know how that algorithm behaves for that test.