Linear/Sequential Search and Binary Search

1. * Here's code for a method that implements a "lazy" sequential (linear) search on a sorted linked list of integers:

```java
boolean lazySearch(Node L, int target) {
    Node ptr = L;
    while (ptr != null && ptr.data < target) {
        ptr = ptr.next;
    }
    if (ptr != null && ptr.data == target) return true;
    return false;
}
```

And here's code for the "regular" sequential search:

```java
boolean search(Node L, int target) {
    for (Node ptr = L; ptr != null; ptr = ptr.next) {
        if (ptr.data == target) return true;
    }
    return false;
}
```

Is the lazy version faster or slower than the regular version? Explain.

2. * An alternative algorithm for searching on an ordered array of size \(N\) works as follows. It divides the array into \(M\) contiguous blocks each of size \(B\). For simplicity you may assume that \(B\) divides \(N\) without remainder. Here is the algorithm to search for a key \(k\):

- Compare \(k\) with the entries \(B, 2B, 3B, \ldots\). In other words, compare \(k\) with the last entry in each block.
- If \(k\) is equal to the entry \(i*B\) for some \(<i\leq M\), then stop with success.
- Otherwise, if \(k > M*B\), stop with failure.

If neither of the above conditions is true, then it must be that \(k < i*B\) for some \(i \leq M\). Then, perform a sequential search on the block of entries from \((i-1)*B + 1\) to \(i*B-1\). Stop with success if \(k\) is equal to some entry in this block, otherwise stop with failure.

What is the worst case number of searches for success?

3. Given the following sequence of integers:

\[3, 9, 2, 15, -5, 18, 7, 5, 8\]

1. What is the average number of comparisons for a successful search assuming all entries are searched with equal probability? Show your work.
2. Suppose the search probabilities for the elements of this list are, respectively:
What is the average number of comparisons for successful search with these search probabilities? Show your work.

3. Rearrange the list entries in a way that would result in the lowest number of comparisons on the average for successful search, given the above probabilities of search. What is this lowest number of comparisons? Show your work.

4. Here’s a Search class that implements a generic sequential search method, i.e. the method accepts an array containing any type of objects.

```java
public class Search {
    public static <T> boolean sequential(ArrayList<T> list, T target) {
        for (int i=0; i < list.length; i++) {
            if (target.equals(list.get(i))) {
                return true;
            }
        }
        return false;
    }
}
```

Now suppose you are writing client code that will use this class/method to sequentially search on a list of Point objects. Here’s a sketch of usage:

```java
public class SomeApp {
    ... // At this point, you should have an ArrayList<Point> points.
    ... // populate the points array list with a bunch of Point objects
    boolean check = Search.sequential(points, new Point(2,3));
    ... // More code here
}
```

Your task is to complete the implementation of the Point class with whatever it takes to make the sequential search for point (2,3) above work correctly. That is, if there is a point in the list that has also x=2 and y=3, then the search must return true, else false. Here’s an outline of the Point class:

```java
public class Point {
    int x, y;
    ... // COMPLETE THIS CLASS
}
```

5. P183, E5.1, Consider binary search on the ordered array used as an example in Section 5.2.2.
(a) List the values according to the number of comparisons it would take to find them. In other words, list all values that can be found with one comparison, all values that can be found with three comparisons, and so on.

(b) What is the average number of comparisons required to find any of the values in the array, assuming they are all searched with equal probabilities. (Hint: Use the answer to the first part, and the formula given in Section 4.2.1)

(c) What would be the average number of comparisons if you were to use sequential search, instead?

6. P184, E 5.4, The binarySearch algorithm of Section 5.2.2 first checks whether the target is equal to the current middle entry, before proceeding to the left or right halves if required. There is another variant of binary search which postpones the equality check until the very end. This version is presented below -- we call it lazy binary search:

```
left <- 0
right <- n-1
while (left < right) do
    mid <- (left+right)/2
    if (t > A[mid]) then
        left <- mid + 1
    else
        right <- mid
    endif
endwhile
if (t == A[left]) then
    display "found at position", left
else
    display "not found"
endif
```

(a) Trace this algorithm on the example array of Section 5.2.2 with 19 as the target. How many comparisons does it make?

(b) Trace the algorithm on this array with 26 as the target. What is the number of comparisons made?

(c) How many comparisons does this algorithm make in general for an array of size n? Give an exact figure, not Big O.

(d) Under what conditions is it preferable to use this version instead of the original binary search version?

7. P341, E10.1, You are given a sorted array of length 11.

(a) Draw a comparison tree for binary search on this array, along the lines of the one drawn in Figure 10.1

(b) What is the worst-case number of comparisons for binary search on this array for a successful search? For a failed search?
(c) What is the average number of comparisons for binary search on this array for a successful search? For a failed search? Assume that all possibilities of success are equally likely. Also that all possibilities of failure are equally likely.

8. P341, E10.2, Repeat each part of the preceding exercise for binary search using the lazy binary search algorithm of E5.4 (Hint: failure and success occur only at leaf nodes. There are no comparison symbols next to any of the internal nodes.) Compare your answer to those obtained in the preceding exercise. Does this tree have more/fewer nodes?

9. Extend the framework of Exercise 4 as follows:
   Introduce the following method in the Search class:

   ```java
   public static <T extends Comparable<T>> boolean binary(ArrayList<T> list, T target) {
     /** COMPLETE THIS METHOD **/}
   ```

   Enhance the Point class by making it implement the interface Comparable<Point> (Comparable<T> is defined in the Java language), and adding the method

   ```java
   public int compareTo(Point p) {
     /** COMPLETE THIS METHOD **/}
   ```

   Two points are compared based on their distances from the origin (0,0). That is, they are "equal" if their distances from the origin are equal, one is "greater" than the other if its distance from the origin is greater than that of the other's.