Reflective Questions Project 6 *Thanapoom Phatthanaphan*

1) Explain why it is better to use a binary search when searching sorted data. Use the code provided in the book and Figure 11-7 to show how to implement a binary search function.

```
def binarySearch(target, lyst):
    """Returns the position of the target item if found,
    or -1 otherwise."""
    left = 0
    right = len(lyst) - 1
    while left <= right:
        midpoint = (left + right) // 2
        if target == lyst[midpoint]:
            return midpoint
        elif target < lyst[midpoint]:
            right = midpoint - 1         # Search to left
        else:
            left = midpoint + 1          # Search to right
    return -1
```

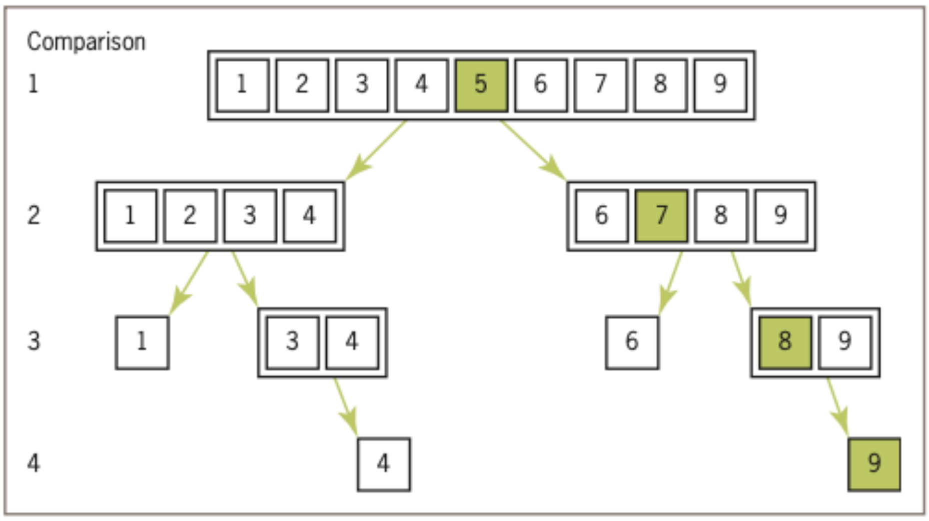

Figure 11-7 The items of a list visited during a binary search for 10

Solution:

The code and the figure above show how the binary search function works. As the example above, we would like to search for the target number from the range numbers 1 to 10 (lyst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]). Using the sequential search method would require to check 10 comparisons in case that the target number is the most right number. More efficiently, we can use the binary search method for less times of comparisons. For the target item 10, the binary search requires only 4 comparisons which is less than a linear search and it will be much less with the larger data size.

The concept of the binary search function is to search for a midpoint of the range that contains the target number as the formula shown above (midpoint = (left + right) // 2). This can let the program ignores to search the numbers in range that does not contain the target number which makes the program runs faster. That is why the binary search is more efficient than sequential search.

2) Briefly explain how the algorithm can be improved, but note that even with this improvement, the average case is still O(*n*2).

Solution:

Some of the ways to improve the algorithm can make the program run faster but the time complexity of the average case is still O(n2). They are basic sort algorithms which include Selection sort, Bubble sort and insertion sort. The selection sort is the strategy to search the entire list for the position of the smallest item and swap with the first position, then it repeats this process and swap the second smallest item with the second position and the list is sorted when the algorithm reaches the last position in this process. The bubble sort starts at the beginning of the list and compare pairs of data items as it moves down to the end. Each time the items in the pair are out of order, the algorithm swaps them. The algorithm then repeats the process from the beginning of the list and goes to the next-to-last item, and so on, until it begins with the last item. At that point, the list is sorted. The insertion sort attempts to exploit the partial ordering of the list in a different way. This strategy consists of two loops. The outer loop traverses the positions from 1 to *n* – 1. For each position *i* in this loop, you save the item and start the inner loop at position *i* – 1. For each position *j* in this loop, you move the item to position *j* 1 1 until you find the insertion point for the saved (*i*th) item.

3) Briefly explain how *memoization* can be used to improve the efficiency of recursive functions that are called repeatedly with the same arguments.

Solution:

This technique, memoization, can reduce the executing time of the program by storing a value for an argument used with the function in the dictionary and simply return that value if the program finds the same argument already has a value in the dictionary without computing that argument. But if the dictionary does not have that argument, the computation proceeds, and that argument and value are added to the dictionary afterward.