Practice Questions on Iteration and Decomposition (Week 4)

Tutorial Questions

1. The following sub-questions concern the array below:

```
x = [8, 9, 11, 16, 32, 37, 43, 45, 52, 55, 58, 61, 63, 66, 68, 82, 95, 99]
```

   a) How many elements will be checked by linear search and by binary search respectively to find the number 8?
   b) How many elements will be checked by linear search and by binary search respectively to find the number 16?
   c) How many elements will be checked by linear search and by binary search respectively to find the number 50?

2. The linear search algorithm (shown below) assumes that each value occurs only once in the array being searched.

```
01  def search(a, k):
02      i = 0
03      while i < len(a):
04          if a[i] == k:
05              return i
06          i = i + 1
07      return -1
```

   Suppose that the array **a** may contain multiple occurrences of the same value.
   a) Modify the above algorithm such that it returns the number of occurrences of the key **k** being searched.
   b) Modify the above algorithm such that it returns the index of the second occurrence of the key **k** if it exists, and **None** otherwise.
   c) Modify the above algorithm such that it returns the index of the last occurrence of the key **k** if it exists, and **None** otherwise.

3. The following algorithm implements insertion sort in ascending order.

```
01  def isort(list):
02    if list.size <= 1:
03      return list
04
05    for i in range(1, len(list)):
06      value = list[i]
07      position = i
08
09      while (position > 0) and value < list[position-1]:
10        position = position-1
11
12      while (i > position):
13        list[i] = list[i-1]
14        i = i - 1
15      list[position] = value
16
17    return list
```

   a) Modify the above algorithm to sort in descending order.
   b) What is the best-case complexity of this modified algorithm, and when does this happen?
   c) What is the worst-case complexity of this modified algorithm, and when does this happen?

4. When merge sort is applied to an array **x**, what is the content of the array **x** just before the final merge step?
   a) When `x = [96, 80, 88, 34, 47, 65, 52, 73]`
   b) When `x = [62, 77, 18, 48, 64, 45, 34, 88, 82, 87, 17]`
   c) When `x = [27, 92, 93, 91, 35, 22, 31]`

5. Let us denote **A** to be a sorted array containing **N** elements, e.g.
   A = `[6, 19, 30, 31, 33, 47, 57, 60].`
   Let us further denote $\mathbf{A_k}$ to be a right-ward shift by `k` steps. For instance, we have
   $\mathbf{A_1}$ = `[60, 6, 19, 30, 31, 33, 47, 57]` and
   $\mathbf{A_3}$ = `[47, 57, 60, 6, 19, 30, 31, 33].`

   Design an algorithm (pseudo-code) to find the maximum element in $\mathbf{A_k}$ in the following conditions:
   a) When the value of **k** is known, and the algorithm should have O(1) complexity.
   b) When the value of **k** is unknown, and the algorithm should have O(N) complexity.
      Hint: linear search
   c) When the value of **k** is unknown, and the algorithm should have O(log N) complexity.
      Hint: binary search

6. You are given as input an unsorted array **B**, containing **n** distinct integers (no duplicates) in the range from 1 to (n+1), whereby exactly one integer in this range is missing from the array. The objective is to determine which integer is missing. For example, for the array **B** = [2, 3, 5, 6, 1, 7], we have **n** = 6, and the range is from 1 to 7. The missing integer is 4.
   Provide an algorithm (in pseudo code or Python code) to find the missing integer in an input array **B**. The lower the complexity of your algorithm is, the higher your marks will be.

Extra Practice Questions

7. For each of the array below, which sorting algorithm: insertion sort or merge sort, will make fewer comparisons?

   a) When `x = [2, 11, 26, 39, 58, 66, 74, 85]`
   b) When `x = [27, 76, 43, 97, 55, 41, 2, 91]`
   c) When `x = [91, 84, 77, 65, 64, 39, 22, 18]`

8. When merge sort is applied to an array of `N` elements:

   a) What is the best-case complexity?
   b) What is the worst-case complexity?
   c) Will merge sort make a smaller, the same, or a greater number of comparisons when sorting an array that is already sorted, as opposed to a random unsorted array?

9. When merge sort is applied to an array **x**, what is the content of the array **x** just before the final merge step?

   a) When `x = [48, 82, 51, 67, 65, 25, 45, 2, 36]`
   b) When `x = [8, 12, 55, 89, 54, 22, 95, 25, 75, 59, 29, 18, 79]`
   c) When `x = [69, 15, 21, 95, 43, 80, 42, 79, 73, 33, 81, 11, 67, 52, 60]`

10. Binary search systematically decomposes the array being searched into two sub-arrays. Ternary search systematically decomposes the array into three sub-arrays. N-ary search decomposes the array into N sub-arrays. If we keep increasing number of sub-arrays from two to three, four, and five and so on, will the resulting N-ary search algorithm be increasingly more and more efficient? Explain why.

11. Suppose you are given a series of **N** numbers as input, where **N** is even. Your objective is to derive pairs of numbers, such that the maximum sum among all pairs will be minimized.

    For example, suppose the numbers are 88, 48, 78, and 73.
    - One pairing solution is (88, 48) and (78, 73) with sums 88 + 48 = 136 and 78 + 73 = 151 respectively. The maximum sum is 151.
    - Another pairing solution is (88, 78) and (48, 73) with sums to 166 and 121 respectively. The maximum sum is 166.
    - The last pairing solution is (88, 73) and (48, 78) with sums to 161 and 126 respectively. The maximum sum is 161.

    The best pairing solution is the one with the smallest maximum sum, i.e., the first pairing solution. Design an algorithm (pseudo-code) to find the best pairing solution, and derive the time complexity of your algorithm. The lower the complexity of your algorithm is, the higher your marks will be.

12. You are given **k** arrays [A_1, A_2, A_3…A_k], where each array A_i has **n** sorted integers. For example, if **k** = 4 and **n** = 3, you have 4 sorted arrays of 3 integers each. Here is a possible set of arrays:
    A_1 = [1, 3, 5], A_2 = [4, 9, 10], A_3 = [2, 6, 7] and A_4 = [8, 12, 15]
    The objective is to combine these **k** input arrays into a single sorted array. The expected output is the combined sorted array, i.e. [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15].

    a) Provide an algorithm (in pseudo code or Python code) to perform this task.
    b) What is the Big O time complexity of your proposed algorithm in terms of **k** and **n**?

~End