SWE-1202 Data Structures and Algorithms

Mid Semester Test

Date: 25th August 2022 | Time: 0900 - 1300 hours

Instructions

- Attempt all questions individually. It will help you!
- Copying will be heavily penalized

This test has 5 questions, for a total of 100 points

1. (a) (7 points) Write down the formal definition of an algorithm.

(b) (3 points) With an example, briefly explain the difference between an Abstract Data Type (ADT) and a data structure

(c) Why do we need

i. (1 point) Data Structures

.....

.....

ii. (2 points) Algorithms

.....

.....

.....

.....

(d) (6 points) Fill in the blanks for INSERTION-SORT algorithm that sorts elements in non-decreasing order.

Algorithm 1 : Given a sequence of n elements $\langle a_1, a_2, \ldots, a_n \rangle$ as Input instance (I), return a permutation(reordering) $\langle a_1', a_2', \ldots, a_n' \rangle$ of I such that $a_1' \leq a_2' \leq \ldots \leq a_n'$

```
1: procedure INSERTION-SORT(A)
2:     for j ← _____, to length[A] do
3:         key ← A[j]
4:         i ← j − 1                ▷ Insert A[j] into sorted sequence A[1...j − 1]
5:         while _____ and _____ do
6:             A[i + 1] ← _____
7:             i ← j − 1
8:         A[_____] ← _____
```

(e) (5 points) Using figure 1 as a model, illustrate the operation of insertion sort on the array $\langle 31, 29, 59, 26, 29, 58 \rangle$

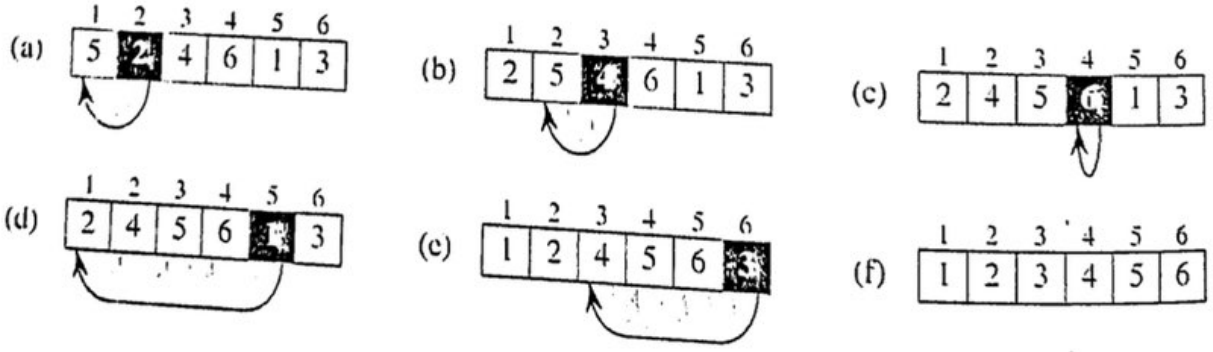


Figure 1: The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$

(f) (1 point) Rewrite your answer to (part d) such that elements are sorted in non-decreasing order.

.....

(g) (3 points) Show that the time complexity for INSERTION-SOTRT is $\mathcal{O}(n^2)$ i.e. quadratic.

.....

.....

.....

.....

.....

.....

.....

.....

(h) (3 points) Express the function $f(n) = \frac{n^3}{1000} - 100n^2 - 100n + 3$ in terms of $\mathcal{O}$ notation while stating briefly the reasoning behind your answer.

.....

.....

.....

2. (a) (8 points) Using the list $\langle 3, 8, 12, 34, 54, 84, 91, 110 \rangle$ compute the number of comparisons for each of the following sorting algorithms. (Note that in this problem we are sorting a list that is already sorted.)

Sort Algorithm	# of Comparisons
Selection sort	
Insertion Sort	
Quick sort	
Merge sort	

Table 1: Fill in this table while showing all your working.

(b) (4 points) Explain the reasons for the answers obtained in question

2 (part a) above especially if the list is already sorted.

(c) (2 points) Suggest improvements to quick sort and explain why those improvements matter.

(d) Given the following list $\langle 90, 8, 7, 56, 123, 235, 9, 1, 653 \rangle$. Trace the execution (You may need to use Tables or diagrams) for:

i. (2 points) Selection sort

ii. (2 points) Quick sort

iii. (2 points) Merge sort

3. (3 points) Showing your reasoning, what is the order of growth (time complexity) of the following code?

Listing 1: Complexity for nested loops

```
1 int result = 0;
2 for (int i = 1; i <= n; i *= 2)
3   for (int j = 0; j < n; j++)
4 result++;
```

4. (a) (5 points) Draw a UML Diagram for the Stack ADT and describe the semantics of the five operations for the ADT.

(b) (5 points) Using the concept of Interfaces in Java, write down the Stack ADT as described above.

(c) (10 points) Fill in the blanks marked with "XXX" for the StackADT implementation using Arrays called the ArrayStack

Listing 3: ArrayStack

```
1  import jsjf.exceptions.*;
2  import java.util.Arrays;
3
4  /**
5   * An array implementation of a stack in which the bottom of
      the
6   * stack is fixed at index 0.
7   *
8   * @author Java Foundations
9   * @version 4.0
10  */
11 public class ArrayStack<T> implements StackADT<T>
12 {
13   private final static int DEFAULT_CAPACITY = 100;
14
15   private int top;
16   private T[] stack;
17
18   /**
19   * Creates an empty stack using the default capacity.
20   */
21   public ArrayStack()
22   {
23    this(DEFAULT_CAPACITY);
24   }
25
26   /**
27   * Creates an empty stack using the specified capacity.
28   * @param initialCapacity the initial size of the array
29   */
30   public ArrayStack(int initialCapacity)
31   {
32    top = 0;
33    stack = XXX (new Object[initialCapacity]);
34   }
35
36   /**
37   * Adds the specified element to the top of this stack,
       expanding
38   * the capacity of the array if necessary.
39   * @param element generic element to be pushed onto stack
```

```
40      *
41      public void push(T element)
42      {
43       if (size() == stack.length)
44           XXX;
45
46       stack[XXX] = element;
47       top++;
48      }
49
50      /**
51      * Creates a new array to store the contents of this stack
          with
52      * twice the capacity of the old one.
53      */
54      private void expandCapacity()
55      {
56       stack = Arrays.copyOf(stack, stack.length * 2);
57      }
58
59      /**
60      * Removes the element at the top of this stack and returns
          a
61      * reference to it.
62      * @return element removed from top of stack
63      * @throws EmptyCollectionException if stack is empty
64      */
65      public T pop() throws EmptyCollectionException
66      {
67       if (isEmpty())
68           throw new EmptyCollectionException("stack");
69
70       XXX;
71       XXX result = XXX;
72       stack[top] = null;
73
74       return XXX;
75      }
76
77      /**
78      * Returns a reference to the element at the top of this
          stack.
79      * The element is not removed from the stack.
80      * @return element on top of stack
81      * @throws EmptyCollectionException if stack is empty
82      */
```

```
83    public T peek() throws EmptyCollectionException
84    {
85     if (isEmpty())
86         throw new EmptyCollectionException("stack");
87
88     return stack[XXX];
89    }
90
91    /**
92    * Returns true if this stack is empty and false otherwise.
93    * @return true if this stack is empty
94    */
95    public boolean isEmpty()
96    {
97     // To be completed as a Programming Project
98     XXX
99     //return true;  // temp
100   }
101
102   /**
103   * Returns the number of elements in this stack.
104   * @return the number of elements in the stack
105   */
106   public int size()
107   {
108    // To be completed as a Programming Project
109    XXX
110    //return 0;  // temp
111   }
112
113   /**
114   * Returns a string representation of this stack.
115   * @return a string representation of the stack
116   */
117   public String toString()
118   {
119    // To be completed as a Programming Project
120    XXX
121    //return "";  // temp
122   }
123  }
```

Linked-list based over and Array based implementation of any ADT?

.....

.....

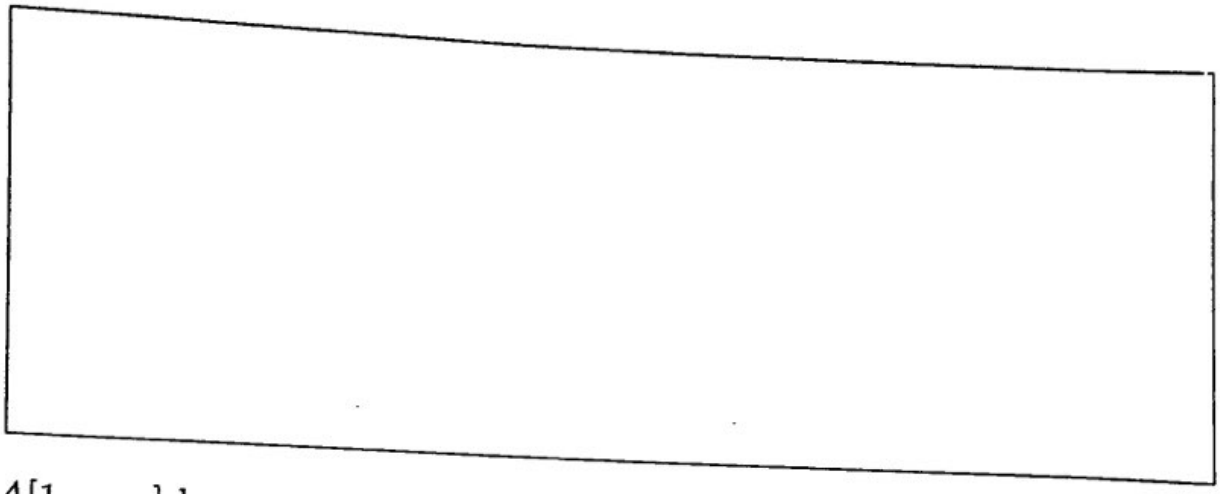
.....

.....

.....

.....

(e) (2 points) Draw an example using the five integers $\langle 12, 23, 1, 45, 9 \rangle$ of how a stack could be used to reverse the order $\langle 9, 45, 1, 23, 12 \rangle$ of these elements.



5. Let $A[1 \ldots n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the $pair(i, j)$ is called an inversion of A

(a) (5 points) List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$

.....

.....

(b) (5 points) What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?

.....

.....

.....

.....

.....

.....

.....

(c) (5 points) What is the relationship between the running time of INSERTION-SORT and the number of inversions in the input array? Justify your answer.

(d) (5 points) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (Hint: modify merge sort.)

(c) (5 points) What is the relationship between the running time of INSERTION-SORT and the number of inversions in the input array? Justify your answer.

(d) (5 points) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (Hint: modify merge sort.)