A. Y. 2022 – 2023

FIRST AND SECOND SEMESTER

PRACTICE PREMIDTERM EXAM 1

CS 3102N – Algorithms and Complexity

Name:	**Christian Stewart**
Program and Year:	**BSCS 3rd Year**
CS 3102N Group #:	**1**
Date:	**22/09/2022**

NOTES:

I. You have **120 minutes** to take the test. Kindly inspect the exam time duration carefully and do not spend too much time on a single question.

II. Each test has different directions. Follow them carefully.

III. Answer each item by typing your answers in the fields and spaces provided.

IV. Always try to write accurately using appropriate and efficient program logic and proper coding syntax. For programming problems, follow the coding conventional rules that are set for the class (e.g. all conditional statements must involve relational operators; break and continue statements shall be only used in switch blocks.).

V. Usage of compilers, calculators, and/or related applications is **STRICTLY NOT ALLOWED**.

VI. Should you have a concern on any of the questions, kindly contact or message the examiner for inspection or clarification.

VII. Answer each question as far as you can. Try not to leave blanks.

VIII. Once you are finished with the exam, save the PDF file with your answers and submit it as an attachment to the examiner's email address (20100215@usc.edu.ph) or through direct message for checking.

I. MULTIPLE CHOICE

DIRECTIONS: Choose the letter of the correct of best answer. Each item is worth 2 points.

B ..... 1. Which of the following is FALSE for selection sort?
[A] Running time is the same for all cases
[B] For an array of N elements, there will always be N exchanges throughout the sort
[C] Running time is the same for all data structure variations
[D] Both A and B
[E] Both B and C

E ..... 2. Among the following algorithms, which can be considered the most efficient for sorting a list of elements where the elements are randomized in position and are far from their sorted positions?
[A] Bubble sort [B] Selection sort [C] Insertion sort [D] Gnome sort [E] Shell sort

C ..... 3. Among the following algorithms, which can be considered the most efficient for sorting a list of elements where only a few of them are left at their unsorted positions?
[A] Bubble sort [B] Selection sort [C] Insertion sort [D] Gnome sort [E] Shell sort

B ..... 4. Why does a bubble sort do a final pass even when the data is in the correct order?
[A] To save the data
[B] It does not recognize that the data is in order until the final pass requires no changes
[C] It needs to do this to put the data back in to a list
[D] None of the choices

C ..... 5. Among the following algorithms, which of them performs best to sort the elements in decreasing order, given an array whose elements are sorted in increasing order?
[A] Bubble sort [B] Selection sort [C] Insertion sort [D] Gnome sort [E] Shell sort

B ..... 6. The auxiliary space of insertion sort is O(1), what does O(1) mean ?
[A] The memory (space) required to process the data is not constant.
[B] It means the amount of extra memory consumed doesn't depend on the input size and contents.
[C] It takes only 1 kb of memory.
[D] It is the speed at which the elements are traversed.

C ..... 7. In proxmap sort, how many additional arrays are created in the entire process, including the array that stores the final sorted positions of the elements?
[A] 2 [B] 3 [C] 4 [D] 5 [E] None of the choices

For items 8 to 10, choose from the following:

[A] Tony Hoare
[B] John von Neumann
[C] J. W. J. Williams
[D] Hamid Sarbazi-Azad
[E] Herman Hollerith
[F] Harold Seward

D ..... 8. Identify the main proponent for **Gnome Sort.**

E ..... 9. Identify the main proponent for **Radix Sort**.

F ..... 10. Identify the main proponent for **Counting Sort.**

II. STRUCTURED RESPONSE

DIRECTIONS: Read carefully and answer the questions correctly. To gain full marks to questions you should express your ideas sensibly and answer with the proper syntax.

1. Complete the table by filling in the correct expressions for the following algorithms.
   - For complexity analysis, Use **n** for number of elements and **k** for the range of the input.
   - For characteristics, consider the following: **S**-stable, **U**-unstable, **CA**-comparison approach, **NCA-**non-comparison approach, **IP**-in-place, **OP**-out-of-place.
   - Indicate the year of discovery for each algorithm.

Name	Time Complexity			Space Complexity	Characteristics	Year Discovered
	Best	Average	Worst			
Bubble	O(n)	O(n^2)	O(n^2)	O(1)	S,CA,IP	---
Selection	O(n^2)	O(n^2)	O(n^2)	O(1)	S,CA,IP	---
Insertion	O(n)	O(n^2)	O(n^2)	O(1)	S,CA,IP	---
Counting	O(n+k)	O(n+k)	O(n+k)	O(k)	S,NCA,OP	1954
Shell	Ɔ(n log n)	(n(logn)^	(n(logn)^	O(1)	U,CA,IP	1959
Radix	O(nk)	O(nk)	O(nk)	O(n+k)	S,NCA,OP	1887
Bucket	O(n+k)	O(n+k)	O(n^2)	O(n)	U,CA,OP	---
Gnome	O(n)	O(n^2)	O(n^2)	O(1)	S,CA,IP	2000

[27]

2. Complete the statements below with the appropriate words or phrases.

   (a) **counting** sort arranges array elements according to the number of times each distinct element appears in the array. It uses an auxiliary array whose size is determined by the **range**

   (b) We can say that an algorithm is **stable** when the order of appearance of the same/duplicated keys maintained in the sorted array like they are initially.

   (c) We can say that an algorithm is **in-place** when there is no extra memory usage except perhaps a small function-call stack or a constant number of instance variables.

   (d) **shell** sort is a simple extension of insertion sort that gains speed by allowing exchanges of array entries that are far apart, to produce partially sorted arrays that can be efficiently sorted, eventually by **insertion** sort.

   (e) Some variations of bucket sort include:

   postman - An algorithm that takes advantage of hierarchical structures of elements;

   histogram - Checks the number of elements that will be in each bucket using a count array;

   proxmap - Uses a "map key" function that preserves a partial ordering on the keys; and

   shuffle - Begins by removing the first 1/8 of the n items to be sorted, sorts them recursively, and puts them in an array.

[10]

3. Given the string **"volkswagen"**: (note: for all sub-questions, to sort the string means the characters are placed in increasing order of their ASCII values). Determine the result of the string:

   a) after the 5th iteration of the outer loop of insertion sort - **klosvwagen** **[2]**

   b) after the 8th swap/exchange of the outer loop of selection sort - **aegklnosvw** **[2]**

   c) after the 8th swap/exchange done in bubble sort, where larger elements are pushed from the lower indices to the upper indices - **olksvagenw** **[2]**

   d) after the 3rd iteration of the outer loop of bubble sort, where smaller elements are pushed from the upper indices to the lower indices - **aegvolkswn** **[2]**

   e) after the 3rd iteration of the outer loop of shell sort, using the original version of the gap sequence - **aegklnosvw** **[2]**

   f) after the 1st iteration of the outer loop of shell sort, using Knuth's increments of the gap sequence - **enagsolkvw** **[1]**

   g) after the 5th swap/exchange done in the gnome sort algorithm - **lkovswagen** **[2]**

4. Given the following list of values: **8, 22, 7, 9, 11, 5, 13**

   a) The number of swaps needed to sort the numbers in ascending order using bubble sort is **10** **[4]**

   b) The number of swaps needed to sort the numbers in ascending order using gnome sort is **10** **[5]**

   c) If counting sort is to be used,

      i) List the values of the count array right after the scanning of elements in the original list:

         - Elements: **0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1** **[1]**

      ii) List the values of the count array after the new array is reproduced in a stable manner:

         - Elements: **0 0 0 0 0 0 1 1 2 3 4 4 5 5 6 6 6 6 6** **[2]**

5. Perform a stable LSD radix sort (base 10) with the values: **121, 432, 564, 23, 1, 45, 788.** Give the following:

   a) Size of the count array: **10** **[1]**

   b) Count array in the first pass/iteration, right before the creation of the auxiliary array:

      - Elements: **0 2 1 1 1 1 0 0 1 0** **[3]**

   c) Auxiliary array created in the first pass/iteration:

      - Elements: **121, 1, 432, 23, 564, 45, 788** **[2]**

   d) Count array in the second pass/iteration, right before the creation of the auxiliary array:

      - Elements: **1 0 2 1 1 0 1 0 1 0** **[2]**

6. What is the technique to make counting sort work for negative numbers?

   max - min + 1

   [3]

7. List down the gap sequence intervals that will be used (in order) in performing a shell sort algorithm in a list of **3,315** elements with the following conditions:

   a) Shell's original sequence: **1657 828 414 207 103 51 25 12 6 3 1** **[2]**

   b) Knuth's sequence: **3280 1093 364 121 40 13 4 1** **[1]**

8. For the **first** pass of the shell sort algorithm with **370** elements, how many elements are scanned in total? Explain how you arrived at your answer.

(0, 364) (1,365) (2, 366) (3,367) (4,368) (5,369)

[3]

9. In this segment of the simplified counting sort, it is argued that this part where elements are placed back from the count array to the sorted array is too inefficient since it has two for loops, making the running time of that segment $O(n^2)$.

```
for(i = 0, k = 0; i < maxValue+1; i++) {
    for(j = 0; j < count[i]; j++, k++) {
        newArr[k] = i;
    }
}
```

a) Do you agree? Explain.

no, because the inner loop does not traverse since it uses the count array index value.

[2]

b) Rewrite that code fragment such that there will be only one for loop. Assume we are sorting a list of integers here. **Make your algorithm stable.**

Use the following variables: `old[] /* old array */, count[] /* count array */, new[] /* new array */.`

```
for(i = size-1; i > 0; i--){
    new[ count[arr[i]] - 1] = old[i];
    count[old[i]]--;
}
```

[6]

10. An argument is made that shell sort has the same time efficiency as insertion sort as the process used by the former is just equivalent to that of the latter, especially during the last few passes. Do you agree? Explain.

Not necessarily because shell sort is faster than insertion sort in cases where elements are farther from their sorted position.

[2]

11. In what way/ is gnome sort similar to the following, and how do their efficiencies differ?

a) Insertion sort

in insertion sort there is a variable that holds the current position of the index that is currently being inserted into its proper position while gnome sort moves forward and backward by incrementing and decrementing the index to be swapped

[2]

b) Bubble sort

All adjacent pairs swap in bubble sort moving forward while the gnome sort moves backward and forward when it detects unsorted pairs.

[2]

12. a) How can bucket sort be made **stable**?

Bucket sort can be made stable if the sorting algorithm used starts inserting from the last bucket and from the last index position of the array.

[3]

b) During what scenarios is a bucket sort effective to use? Also, when does it perform worst?

bucket sort is effective when the range and size of the list is large and distributed randomly. It performs worst when all the values of the list are stored in one bucket - therefore, it is skewed.

[3]

13. Compare the performance of proxmap sort with counting sort because they seem to follow a similar procedure. In what way/s is one more or less efficient than the other? Justify your answer

Proxmap sort is less efficient in terms of space because it uses 4 additional arrays compared to counting sort where it uses 2.

[3]

14. In proxmap sort, there are several auxillary arrays mentioned that are created. Identify and explain the use of each auxiliary array created.

[5]

15. In radix sort, right after creating the count array representing the occurrence of each certain digit, it was mentioned that we need to decrement the value in the first index (index 0) by 1. Why is this significant?

[2]

16. Identify and distinguish between LSD radix sort and MSD radix sort. Is there a difference between the performance of the two?

LSD starts with the least significant digit in the list while MSD starts with the most significant digit. MSD outperforms LSD when it comes to strings but it is rather insignificant when it comes to integers. MSD also brings us closer to the sorted position immediately since it uses the most significant digit.

[2]

17. Provide a scenario where the worst-case time/space complexity (where applicable) of each sorting algorithm would occur. Here, to sort a list of values means that the items are arranged in ascending order of their keys.

a) Bubble sort **reverse sorted**

b) Selection sort **no worst case because time complexity is same in all cases**

c) Counting sort **When interval between values is significantly large**

d) Radix sort **when all elements are of the same digits**

[4]

18. Complete the code on the right to implement shell sort. Fill in the blanks with the right syntax. Be careful of semicolons.

```
void shellSortv2(int array[], int n) {
   int interval, i, j, temp;
   for(interval = 1; interval < n; [B1]);
   // 1, 4, 13, 40, 121, 364, 1093, ...
   for (; [B2]; interval = interval / 3) {
     for ([B3]; i < n; i++) {
        temp = array[i];
        for (j = i; [B4]; [B5]) {
          array[j] = array[ [B6] ];
        }
        array[j] = temp;
     }
   }
}
```

Answers here

B1: `interval = 3 * interval + 1`

B2: `interval > 0`

B3: `i = interval`

B4: `j >= interval && array[j - interval`

B5: `j -= interval`

B6: `j - interval`

[7]

III. PROGRAMMING

DIRECTIONS: Read carefully and give accurately what is asked. Make your code concise, efficient, and readable. All functions should have up to one (1) return statement only.

19. Implement Gnome sort using a doubly linked list data structure defined on the right. Given a list of elements in random order, these are to be sorted in increasing order of the values in the data field. Retain the positions of the given list and instead, have a new list containing the sorted elements then have that list returned to the calling function.

```
typedef struct cell1{
   int data;
   struct cell1* prev;
   struct cell1* next;
}node1, *List1;
```

```
List1 gnomeSort(List1 head) {
    List1 *trav, temp;

    for(trav-&head; *trav)
}
```

[15]

20. Implement LSD Radix sort with base 16 using a singly linked list data structure defined on the right. Given a list of elements in random order, these are to be sorted in increasing order of the values in the data field. Ensure all dynamically allocated memory are retuned accordingly to avoid any memory leaks. Declare your own auxiliary data structures as necessary.

```
typedef struct cell2{
   int data;
   struct cell2* next;
}node2, *List2;
```

```
void radixSort(List2 *head) {
  List2 *trav;
  int max;

  for(trav=head;*trav!=NULL;trav=&(*trav)->next) {
    if(max < (*trav)->data) {
      max = (*trav)->data);
    }
  }


}
```

[15]

"If you're always trying to be normal, you will never know how amazing you can be."
- Maya Angelou -

=== THE END ===

God bless you!

REVIEW YOUR ANSWERS!

Prepared by:
Wayne Dayata