Mr. Sakdipat Ontoum
ID: 61130500229
CSC209 Data Structures

1. Many operations can be performed faster on sorted than on unsorted data. For which of the following operations is this the case?

a. checking whether one word is an anagram of another word, e.g., *plum* and *lump*

b. finding an item with a minimum value

c. computing an average of values

d. finding the middle value (the median)

e. finding the value that appears most frequently in the data

Ans:

A)

No, the examples are the "plum and lump", taking the exact word "plum" if the words holding in plum are arranged in sorted list then it will propose a small advantage, if the arranged values are unsorted then quickly check whether any of the word occurs inside the array of words.

Reason:

- Each word needs to be taken and then that word must be compared.
- After this comparison only the final checking process will complete.
- All these processes are consuming more time, it is slowing the process.
- Hence, "finding anagram for each word" is "Not faster".

B) Yes, the advantages are huge in this case, if the values are sorted arrays.

Reason:

- The most significant things are the concern of the last element in the list are the minimum value or not.
- If the minimum value is found, then the sorting is faster in the sorted content of arrays.
- Hence, "Implementing a minimum valued item" is "Faster for sorted".

C) No, for computing the average it will not be the faster one.

Reason:

- Average calculation steps are very familiar to everyone.
- Initially all the value needs to be added then counting of all the elements finally a division operation needs to be operated for finding the average.
- The Entire process will consume more time hence this will not be the faster one.
- Hence, "Average value computation for all the values" is "not faster".

Mr. Sakdipat Ontoum
ID: 61130500229
CSC209 Data Structures

D) Yes, apparently. If the set of integers contains the odd number of values, then compute the middle index and chose that value.

Reason:

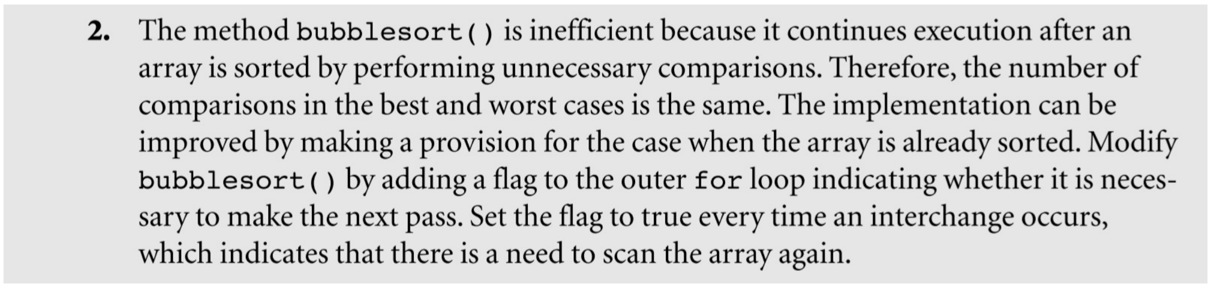
- If the set of integers contains the even number of values, then compute the very closest- middle index and find the average of 2 middle-most values.
- Then this operation can be performed faster.
- Hence, “calculation with the middle values” is “faster”.

E) Yes, it is applied by a modified insert at the time of counter increment for each value just like traversing the list, but only for the number of reasonable bins.

Reason:

- Otherwise, if the data is sorted it is possible to implement a single linear traversal and counting the longest running of the identical values.
- So sorted data is providing the advantage for this case.
- Hence, “Implementing most frequently appeared” is “faster”.

Mr. Sakdipat Ontoum
ID: 61130500229
CSC209 Data Structures



2. The method `bubblesort()` is inefficient because it continues execution after an array is sorted by performing unnecessary comparisons. Therefore, the number of comparisons in the best and worst cases is the same. The implementation can be improved by making a provision for the case when the array is already sorted. Modify `bubblesort()` by adding a flag to the outer `for` loop indicating whether it is necessary to make the next pass. Set the flag to true every time an interchange occurs, which indicates that there is a need to scan the array again.

Ans: Bubble sort continues execution even after an array is sorted. To prevent unnecessary comparisons, we add a Boolean variable say switched and initialize it by True in the beginning. Along with the "for" loop, we add the condition (switched = true) and make it false inside the outer done in the first pass, no more comparisons will be done further, and the program shall exit.

The algorithm after modifying it in the above-mentioned manner will be as follows: -

```
void bubble(int x[] , int n){

int j, pass, hold;

book switched = true;

for(pass = 0 ; pass < n-1 && switched = true ; pass++){

    switched = false;

   for(j = 0 ; j < n-pass-1 ; j++) {

          witched = true;

          hold = x[j];

          x[j] = x[j+1];

          x[j+1] = hold;

       }

    }

}
```