Assignment 3: Median Wars

Background

For the purposes of this assignment, the median of a collection of values is the one that is "in the middle" of the collection. If you arranged the values in increasing order, the median would be the one whose position is midway along the list. If the size of the list is even, use the value on the "high" side of the mid point (the "upper median").

Medians are often found using selection algorithms; the median of **n** items can be found by selecting for item **n/2**. In this assignment, you'll explore median-finding algorithms based on the quickselect and counting select procedures.

Quickselect uses the same partitioning idea as quicksort. After a partition, the item chosen as the pivot will have all smaller items to its left and all larger items to its right, which means it will be in its correct position in the list. If the position of the pivot is not the one you're looking for, the algorithm proceeds by selecting in either the left or right sublists, depending on which side of the pivot position the desired item falls.

```
// return the kth smallest item
int quickSelect(int items[], int first, int last, int k) {
  int pivot = partition(items, first, last);
  if (pivot > first + k) {
    return quickSelect(items, first, pivot, k);
  } else if (pivot < first + k) {
    return quickSelect(items, pivot+1, last, first+k-pivot);
  } else {
    return items[pivot];
  }
}
```

Counting select uses a similar approach to counting sort, where items in the list are used as indexes into an array of counts. Initially the count array contains all zeros. As the list is traversed, the appropriate count values are incremented; when all items have been processed, the value at each index is a count of the number of times an item equal to the index was present in the list. Once the counts have been determined, the **k**th item can be found by beginning at the low value end of the array and accumulating counts until the total exceeds **k**.

```
// return the kth smallest item
// assumes 0 <= item value < MAXITEM
int countingSelect(int items[], int first, int last, int k) {
  int counts[MAXITEM];
  for (int c = 0; c < MAXITEM; c++) {
    counts[c] = 0;
  }
  for (int i = first; i < last; i++) {
    counts[items[i]] += 1;
  }
  int c = 0;
  while (k >= 0) {
    k -= counts[c++];
  }
  return c-1;
}
```

Core Questions (5 marks each)

1. Explain the median-finding algorithms based on quickselect and counting select in a way that would be understandable to an intelligent lay person. You should not use any code (or even pseudo code) in your explanation, but you may need to use general concepts such as "compare", "swap", and "copy", and you'll certainly need to use procedural words such as "if" and "repeat".

   You might find it helpful to consider an algorithm as if it were a game for which you need to define the rules. For example, an easy-to-understand select algorithm (but not a very efficient one) is based on a truncated version of selection sort, where the sort is stopped when the **k**th smallest item is moved into position:

```
// return the kth smallest item
int selectionSelect(int items[], int first, int last, int k) {
   for (int i = first; i <= first+k; i++) {
     int min = i;
     for (int j = i + 1; j < last; j++) {
       if (items[j] < items[min]) min = j;
     }
     swap(items[i], items[min]);
   }
   return items[k];
}
```

   If your task was to describe how you could use this approach to find the median, you could describe the process as if it was a solitaire game played with a deck of cards that contain the values to process.

> **Median-in-the-Middle**
>
> The playing area consists of four regions: foundation, tableau, stock, and discard. Initially, all cards are in the stock.
> Play consists of a number of rounds. To begin a round, place the top card of the stock face up to begin the tableau, then turn over the the next card. If the stock card is smaller than the tableau card, play it on the tableau.; otherwise, discard it. When all of the cards in the stock have been played, move the topmost tableau card to the foundation. Then combine the remaining tableau cards with the discards to form the stock for the next round.
> Continue to play rounds until the number of cards in the foundation is greater than the number of cards in the stock. The median card is then the topmost card in the foundation.

2. Write a set of guidelines for deciding which of the two median-finding algorithms would be most appropriate for a particular situation. Include in your guidelines a description of the advantages and disadvantages of each algorithm, together with an indication as to why those characteristics apply. Your goal should be to provide enough information so that someone not familiar with the details of each algorithm would be able to decide which one is right for them.

Extension Questions (5 marks each)

The running medians of a sequence of values are the medians of successively longer subsequences. Thus the first running median is the median of the first item alone, the second running median is the median of the first 2 items, the third is the median of the first 3 items, and so on. The following example shows the idea:

```
Sequence:         7  9  3  8  0  2  4  8  3  9
Running medians:  7  9  7  8  7  7  4  7  4  7
```

3. Devise an efficient technique for computing the running median of a sequence of values. Note that a technique that is efficient for computing a single median for a sequence may not be the most efficient for computing the running medians. For example, simply using the quickselect process to compute successive running medians will not be very efficient because quickselect must start again each time; it can't take advantage of the computation done in the previous select, even though most of the values are the same.

   Analyse the running cost of your algorithm, and describe it in sufficient detail that it could be implemented by someone familiar with C++ programming.

4. Implement your running medians algorithm, together with a test program. Instrument your code to measure its running cost, and gather experimental data that you can use to compare the actual running cost with your theoretical analysis.

   Write a brief report that describes your experimental setup and presents your findings. You'll need to consider how best to present your results. You'll certainly want to tabulate the data, but you might also find it helpful to plot it as well. And because running costs will be heavily dependent on the size of the sequence, you might find it useful to normalise the costs first.

Assessment

Pairing

This assignment must be done individually.

Scoring

Submit written material as PDF reports to the appropriate handins on FLO. The "core" score will be based on your answers to the 2 "core" questions, and the "extension" score will be based on your answers to the 2 "extension" questions.

Your answer for each question should be between 300 and 600 words (half to 1 typed page), excluding code listings (where appropriate), which should be as an appendix in your report. Your submission should conform to accepted practices for academic writing. Of course, you must give appropriate acknowledgement to any material that you use or reference.