Assignment specifications

Empirical Comparison of Two Algorithms

Due 21 May

40%

Grouped

Sample assignment provided

Compile code:  
gcc Assignment2.c -std=gnu99 -lncurses -o Assignment2

Summary

Two algorithms that complete the same functions but have different efficiencies

Analyse under the same conditions

Count the number of basic operations performed, measure execution time

Tasks

1. **Algorithm Understanding**
   1. Briefly describe the two algorithms

For assignment two of CAB301 there are two algorithms that both complete the same task; to identify the median value from within a given array. The two algorithm are to be identified as "BruteForceMedian" and "EfficientMedian". These two algorithms are to be compared to one another in terms of their efficiency. The most efficient algorithm is described as the algorithm that performs the least number of basic operations, performs in the least amount of time but that also returns the correct value. The BruteForceMedian is a single algorithm that requires the array 'A' and returns the median value of 'A'. The "EfficientMedian" is comprised of three subfunctions; 'Select' , 'Partition' and 'Swap'. An additional algorithm 'Print\_Array' is also used to print the array A for purposes of communication.

'BruteForceMedian' operates by the use of three variables 'numsmaller', 'numequal', K and two for loops where one is nested within the other. Whee k is equal to the array length / 2. Both for loops perform a number of cycles equal to the length of A. Within the nested for loop a comparison of the next element with the previous is performed to identify the smallest.

"A[j] < A[i]"

If this returns true the 'numsaller' variable is increased

If the next value is found to be equal to the previous the 'numequal' variable is creased.

The last comparison to be completed is that where:

"if( (numsmaller < k) && k <= (numsmaller + numequal) )"

If this returns true the A[I] is returns; the median.

The second algorithm operates by the use of four separate functions and operate in order of 'median','select', 'partition' and 'swap'. The functions 'median' checks if the length of array A is 0, if true A[0]; the first value and only value is returned as the median. If the array length is not 1 the 'Select' function is then run:

The Select function takes the variables SELECT(A,l,m,h) and is passes as Select(A, 0,[n/2], n − 1)

The 'select' function initially stores a integer variable 'pos' that's defined by running the function 'Partition':

Partition(A, l, h)

Select then compares the variable pos and m. If pos is equal to M, the value in the array located as A[pos] is returned. If pos is larger than m, the Select function is rerun with:

Select(A, l, m, pos − 1)

If pos is smaller than m, the select function is rerun with:

Select(A, pos + 1, m, h)

The 'Partition' function operates by by having Partition(A[0..n − 1], l, h) passed to it.

Partition initially storing two variables:

Pivotval = A[l] and Pivotloc = l

A for loop is executed where that runs for a number of cycles equal to h

For each cycles j of the for loop, if A[j] < pivotval the value of pivotloc is increased by one and the swap function is called to swap values at A[pivotloc] and A[j]. Once the for loop is completed the swap function is again called and the values at A[l] and A[pivotloc] are swapped.

After the swap is completed the variable pivotloc is returned to the 'select' function.

The 'swap' function operates by having swap(int first,int second) passed to it.

Swap initially stores a variable:

Temp = first.

Then the passed values 'first' is set to be equal to ' second' and the second passed value' second' is set to be equal to the function variable' temp'.

1. **Common basis for meaningful comparison (basic operations and size of inputs)**
   1. **Reasoning for basic operation selection**

Basic operation in the case of the two algorithms is where one value is compared to another.

Algorithm 1: Brute Force

There are three points/ cases where values are compared within the first algorithm.

These are considered to be the basic operations completed.

For a single cycle of the algorithm a maximum of two of the basic operations are able to occur. This is because two of the basic operations are within an if, else query.

Although the third basic operation is not a comparison to the Array of data it is a comparison of data derived from the array, and the result of does pass the median value of the array.

Apart from the three comparisons there is little content within the algorithm apart from two for loops. The selection of basic operation is logical and justified.

For algorithm 2 the basic operations

There are six points/ cases where values are compared within the second algorithm. As the second algorithm is a compilation of three other functions the basic operations are also spread among them.

Three within the select function, where only one can be triggered per calling of the select function as they basic operations are within 'if', 'else if' and 'else if' comparisons of the.

Other than these three queries, there is little content within the Select function. The selection of basic operation is logical and justified.

Two within the partition function, where in the best case/ minimal case only one of the basic operations are triggered; that is the for loop and nested if statement do not return true. However in the more common case both basic operations occur. In both cases of basic operation the swap function is called where the next value is swap places with the previous value

For the two algorithms wherever a comparison occurs between values within the array A, the basic operation counter respectively increases.

1. **Reasoning for 'Problem size' selection**

Length of array & range of values

Both are relative to one and other

When selecting Array constraints for the first algorithm, the array length is the only variable to be have affect on the number of basic operations to occur. Because the first algorithm does not take into consideration value range and ordering.

The second algorithms number of basic operations completed and associated processing time is subject to array length, value range and ordering.

Characteristics, optimising

If the array length is small

If the spread of values is small the median is

1. **Predicted theoretical time efficiency; the average case efficiency.**

Pivot sort algorithm and brute force

1. **Methodology behind experimentation**
   1. **Computing Environment**

Coded in C, using Notepad++, compiled in Cygwin

1. **Implementation of algorithm. Correspondence between algorithm description and the coded implementation**

File 1: exact copy and implementation of provided pseudo code

1. **Show/ explain how the coded algorithms work correctly. Testing.**

Testing: short arrays both order and un order where the median is known

Show both algorithms results

1. **How test data for experiments was produced. Same test data should be applied to both algorithms.**

The random filling array generated examples a,b,c

Tested a,b,c on both algorithms

Ordered filling array, random filing array

1. **How basic operations are counted. Show relevant code; 'counter' variable.**

Code screenshots

1. **How execution time is counted. Show relevant code; Start and stop timer blocks. For small tests it's best to complete them multiple times and then find the average time in total.**

Located within the main() function

Complete the same array multiple times to and find average

1. **Describe results of the experiment**
   1. **Briefly explain how the program works correctly. Testing**
   2. **Detailed results of comparing number of basic operations performed by the two algorithms for different size of inputs. For the two algorithms plot together**
   3. **Detailed results of comparing amount of time takes for the two algorithms for different inputs sizes.**
2. **Write report**
3. **Submit**
   1. **Electronic copy of repot**
   2. **Complete source code for implemented algorithm**
   3. **Complete source code of testing procedures**