CAB302 - Assignment 2

Luke Josh, Jason Queen

May 12, 2016

Contents

1 Summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In et mauris felis. Etiam viverra molestie euismod. Nullam finibus nisl et mollis molestie. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec viverra lacinia ligula tristique convallis. Sed eget erat leo. Phasellus convallis blandit rhoncus. Donec tellus sapien, fringilla auctor purus sit amet, blandit mollis eros. Nulla arcu turpis, malesuada a odio mollis, sollicitudin varius elit. Vestibulum euismod dictum fermentum. Morbi consectetur bibendum quam, ac dictum velit hendrerit posuere. Etiam vel massa sit amet dui porta vehicula id ac diam.

2 Description of the Algorithms

Duis scelerisque risus at urna efficitur, id laoreet dolor eleifend. Aenean sed nulla quis quam iaculis congue. Vivamus posuere dui in ornare posuere. Integer accumsan, diam ut pharetra vulputate, sapien neque consectetur nisl, in gravida neque lacus nec quam. Nullam sit amet pulvinar quam, sed tristique ante. Nunc eleifend orci non orci fringilla viverra quis eget libero. Pellentesque in metus non augue sodales tempor. Vivamus maximus malesuada ligula vel interdum. Nulla pretium lectus vel nisl faucibus, vitae pretium ante venenatis.

2.1 Brute Force

The brute force median algorithm works by manually checking each value of the array, and seeing if it's position in a sorted array is in the median position of $|k/2|$

2.1.1 The Algorithm

Algorithm 1 Brute Force Median

```
function BruteForceMedian(A[0..n − 1])
    k ← ‖n/2‖
    for i ← 1 to n − 1 do
        numsmaller ← 0
        numeqal ← 0
        for j ← 0 to n − 1 do
            if A[j] < A[i] then
                numsmaller ← numsmaller + 1
            else
                if A[j] = A[i] then
                    numequal ← numequal + 1
                end if
            end if
        end for
        if numsmaller < k and k ≤ (numsmaller + numequal) then
            return A[i]
        end if
    end for
end function
```

2.2 Johnsonbaugh and Schaefers Algorithm

Words

2.2.1 The Algorithm

Algorithm 2 Selecion Median

```
function MEDIAN(A[0..n − 1)
    if n = 1 then
        return A[0]
    else
        Select(A, 0, ⌊n/2⌋, n − 1)
    end if
end function

function SELECT(A[0..n − 1], l, m, h)
    pos ← Partition(A, l, h)
    if pos = m then
        return A[pos]
    end if
    if pos > m then
        return Select(A, l, m, pos − 1)
    end if
    if pos < m then
        return Select(A, pos + 1, m, h)
    end if
end function

function PARTITION(A[0..n − 1], l, h)
    pivotval ← A[l]
    pivotloc ← l
    for j ← l + 1 to h do
        if A[j] < pivotval then
            pivotloc ← pivotloc + 1
            swap(A[pivotloc], A[j])
        end if
        swap(A[l], A[pivotloc])
    end for
    return pivotloc
end function
```

3 Theoretical Analysis of the Algorithms

3.1 Brute Force Median

3.1.1 Choice of Basic Operations

The operation that best defines the complexity and running time of the brute force median algorithm is the comparison $A[j] < A[i]$. This comparison operation is performed more than any other operation in the algorithm - a minimum of $n - 1$ times, and a maximum of $(n - 1)^2$ times.

3.1.2 Choice of Problem Size

Words

3.1.3 Average Case Efficiency

The average case efficiency of the algorithm can be derived by considering then number of operations required to determine the median of an arbitrarily sized array. Consider an array $A = [a_1, a_2, ..., a_n]$ with it's median value placed at position k. The algorithm must check each element in the array up to and including $A[k]$ against each element in the array, including itself. Thus, the algorithm must perform $k \cdot n$ comparisons to determine that $A[k]$ is the median value of the array.

To determine the average number of operations for an array of size n, we must consider the pairity of the size of the array, as the algorithm chooses the value to the left of the midpoint when there are an even number of elements. We will first assume that each value in the array has an equivalent chance of being the median. In this case, the average case number of operations to determine the median is average number of operations for each $M = A[k]\ \forall k \in [0, 1, ..., n]$:

$$c_{average} = \frac{\sum_{j=1}^{n} j \cdot n}{n} \tag{1}$$

$$c_{average} = \sum_{j=1}^{n} j \tag{2}$$

$$c_{average} = \frac{n^2 + n}{2} \tag{3}$$

To account for the issue of pairty, consider two sorted arrays, $A = [a_1, a_2, ..., a_n]$ and $B = [a_1, a_2, ..., a_{n+1}$, where n is odd. In each of these arrays, the median value will be the same, $M = a_{floor(\frac{n}{2})}$, and will have to check the same number of elements to find it, however, in B, each element is checked against $n + 1$ other elements, as opposed to A's n other elements. We can use this to restate the above value for the average number of comparisons as $c_{oddaverage} = \frac{n^2+n}{2}$, and conversely, $c_{evenaverage} = \frac{n^2}{2}$. These two statements can be combined to give the true average number of comparisons for an arbitrarily sized array as:

$$c_{average} = \frac{n^2 + (n \mod 2) \cdot n}{2} \tag{4}$$

Which gives the algorithm an average case complexity of $\Theta(n^2)$.

3.2 Selection Median

3.2.1 Choice of Basic Operations

Words

3.2.2 Choice of Problem Size

Words

3.2.3 Average Case Efficiency

Words

4 Methodology, Tools and Techniques

4.1 Programming Environment

4.2 Implementation of the Algorithms

4.3 Generating Test Data and Running the Experiments

5 Experimental Results

5.1 Functional Testing

5.2 Average-Case Number of Basic Operations for an Item in the Set

5.3 Average-Case Number of Basic Operations for an Item not in the Set

5.4 Average-Case Execution Time for an Item in the Set

5.5 Average-Case Execution Time for an Item not in the Set