Lab 5 Selection

Michael Morikawa

April 9, 2020

Lab Questions

Question 1 Provide some good reasons that you might not want to sort the data in order to determine the median value.
You do not need to sort the data to get the median because it does extra work and for larger data sets it will be slow. You could partially sort but again for large data sets it would be a lot of work.

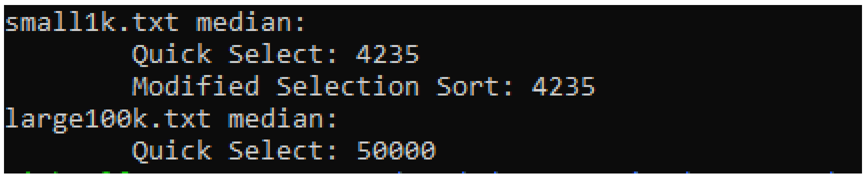
Question 2 Explain how the prune-and-search design pattern works. Is recursion always necessary? Explain.
The prune-and-search design patter works by removing/pruning away elements from the collection that we are working until we reach a set size. Once the size is reached the problem is solved using brute force. Recursion is not always necessary; you can sometimes iteratively reduce the elements you are working with. For example you do an iterative binary search which would fall under this category of algorithms.

Source Code

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <time.h>

//returns the kth smallest element of the array
int kSelectionSort(std::vector<int> arr, int k);
int quickSelect(std::vector<int> S, int k);
std::vector<int> randomizeVector(int size);
void loadFile(const char path[], std::vector<int> &S)
{
    std::ifstream infile;
    infile.open(path);
    int temp;
    if (infile.fail())
    {
        std::cout << "Couldn't find " << path;
        return;
    }
    while (true)
    {
        if (infile.eof())
        {
            break;
        }
        infile >> temp;
        S.push_back(temp);
    }
    infile.close();
}

int main()
{
    std::vector<int> small1k;
    std::vector<int> large100k;
    loadFile("docs/large100k.txt", large100k);
    loadFile("docs/small1k.txt", small1k);
    int kSmall1k, kLarge100k;
    kSmall1k = quickSelect(small1k, small1k.size() / 2);

    std::cout
        << "small1k.txt median:\n"
        << "\tQuick Select: " << kSmall1k << "\n";
    kSmall1k = kSelectionSort(small1k, small1k.size() / 2);
    std::cout << "\tModified Selection Sort: " << kSmall1k << "\n";

    kLarge100k = quickSelect(large100k, large100k.size() / 2);

    std::cout
        << "large100k.txt median:\n"
        << "\tQuick Select: " << kLarge100k << "\n";
}

int kSelectionSort(std::vector<int> arr, int k)
{
    int key;
    int j;
    int minIndex;
    for (int i = 0; i < k; i++)
    {
        minIndex = i;
        for (j = i + 1; j < arr.size(); j++)
        {
            if (arr[j] < arr[minIndex])
            {
                minIndex = j;
            }
        }
        std::swap(arr[minIndex], arr[i]);
    }
    return arr[k - 1];
}

int quickSelect(std::vector<int> S, int k)
{
    std::srand(time(NULL));
    std::vector<int> greater, lesser, equal;
    if (S.size() == 1)
    {
        return S[0];
    }
    int pivot = S[rand() % S.size()];
    for (int i = 0; i < S.size(); i++)
    {
        if (S[i] < pivot)
        {
            lesser.push_back(S[i]);
        }
        else if (S[i] > pivot)
        {
            greater.push_back(S[i]);
        }
        else
        {
            equal.push_back(S[i]);
        }
    }
    if (k <= lesser.size())
    {
        return quickSelect(lesser, k);
    }
    else if (k <= (lesser.size() + equal.size()))
    {
        return pivot;
    }
    else
    {
        return quickSelect(greater, k - lesser.size() - equal.size());
    }
}

std::vector<int> randomizeVector(int size)
{
    std::srand(time(NULL));
    std::vector<int> randVec;
    for (int i = 0; i < size; i++)
    {
        randVec.push_back(std::rand() % 1000);
    }

    return randVec;
}
```

Output



```
small1k.txt median:
        Quick Select: 4235
        Modified Selection Sort: 4235
large100k.txt median:
        Quick Select: 50000
```