計算機演算法作業 1
分析 sorting 時間

班級:資訊三甲

學號:D0617735

姓名:霍湛軒

規則建立說明

Bubble Sort:

第一個數字都要要跟第二個個數字比較，較大的跟後面繼續比，直到把最大的數字放到最後，第二大的放到倒數第二個位置。

Insertion Sort:

每次從數列的最後插曲數字並向前比較及交換，放到適合的位置。

Selection Sort:

每一次從數列中找出最小的數值並與數列的第一個數字交換直到最後一個數字。

Quick Sort:

先選擇一個數列中最後的一個數字作為pivot, 掃描整個數列把小於pivot的數字放在數列的左半邊，大於pivot的數字放在數列的右手邊，並把pivot放到中間。

Minimal Heap Sort:

插入所有數字並調整成Minimal Heap，每次delete root並重新調成Minimal Heap，直到heap清空。

程式碼

Bubble Sort

```
int* bubbleSort(int* data, int size)
{

    for (int i = size - 1; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            if (data[j] > data[j + 1])
            {
                int tmp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = tmp;
            }
        }
    }
    return data;
}
```

Insertion Sort

```
int* insertionSort(int* data, int size)
{

    for (int i = 1; i < size; i++)
    {
        for (int j = i; j > 0; j--)
        {
            if (data[j] < data[j - 1])
            {
                int tmp = data[j];
                data[j] = data[j - 1];
                data[j - 1] = tmp;
            }
        }
    }
    return data;
}
```

Selection Sort

```
int* selectionSort(int* data, int size)
{

    for (int i = 0; i < size; i++)
    {
        int minIndex = i;
        for (int j = i + 1; j < size; j++)
        {
            if (data[j] < data[minIndex])
            {
                minIndex = j;
            }
        }
        if (minIndex != i)
        {
            int tmp = data[i];
            data[i] = data[minIndex];
            data[minIndex] = tmp;
        }
    }
    return data;
}
```

Quick Sort

```
int* quickSort(int* data, int left, int right)
{
    if(left < right){
        int i=left - 1, piovt = data[right], tmp;
        for(int j=left; j<right; j++){
            if(data[j] < piovt){
                i++;
                tmp = data[i];
                data[i] = data[j];
                data[j] = tmp;
            }

        }
        i++;
        tmp = data[i];
        data[i] = data[right];
        data[right] = tmp;
        quickSort(data, left, i - 1);  // 對左子串列進行快速排序
        quickSort(data, i+1, right);   // 對右子串列進行快速排序
    }
    return data;
}
```

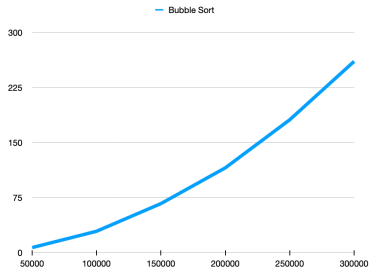
Min Heap Sort

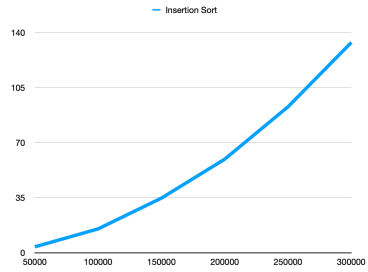
```
void minHeapify(int* data, int root, int size)
{

        int left = 2 * root, right = 2 * root + 1, min = root;
        if (left < size && data[left] < data[root])
        {
            min = left;
        }
        if (right < size && data[right] < data[min])
        {
            min = right;
        }
        if (min != root)
        {
            int tmp = data[min];
            data[min] = data[root];
            data[root] = tmp;
            minHeapify(data, min, size);
        }
}

int* buildMinHeap(int* data, int size)
{

    for (int i = size / 2; i >= 1; i--)
    {
        minHeapify(data, i, size);
    }
    return data;
}

int* minHeapSort(int* data, int size)
{
    data = buildMinHeap(data, size);
    int* sorted = (int*)malloc(size*sizeof(int));
    int j = 0;
```

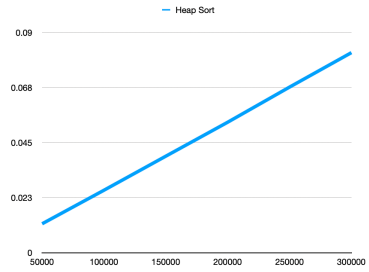
```
    for (int i = size - 1; i >= 1; i--)
    {
        sorted[j++] = data[1];
        data[1] = data[i];
        minHeapify(data, 1, i);
    }
    return sorted;
}
```

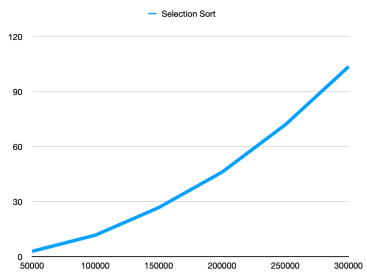
執行結果

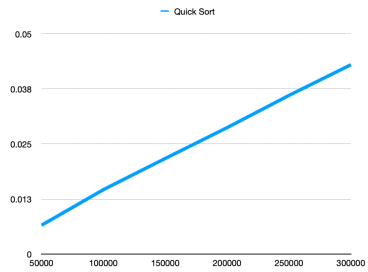
	Bubble Sort	Insertion Sort	Selection Sort	Quick Sort	Heap Sort
50000	6.9376	3.71348	2.8992	0.00656	0.0118
100000	29.15304	15.13296	11.72508	0.01464	0.02552
150000	66.86028	34.69852	26.73496	0.02172	0.03944
200000	115.57596	59.46176	46.05052	0.02872	0.05332
250000	181.07004	92.7774	71.902	0.03604	0.06768
300000	260.59428	133.65648	103.55488	0.043	0.08176

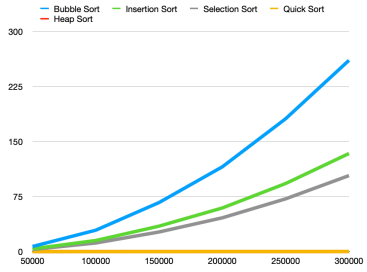












討論：

從圖青上可以看到各個排序演算法所花的時間符合公式的計算。但可看到Bubble Sort, Insertion Sort及Selection Sort的理想時間雖然都是n^2但真正的時間卻相差很遠，訝明BigO只能把演算法劃分等級，但同等級的卻可以相差很大。

心得：

一開始使用MacOS進行測試，發現Quick Sort的執行時間不符合預期，發現在Windows下執行的時間較快，但仍然無法達到預期，因此分別用G++及Clang進行compile，G++ 的時間較快，但執行時間還是太慢，發現vector會大幅令qucik sort的時間上升但其他sorting並沒有，最後把所有的vector換成int pionter。