國立臺南大學資訊工程學系

資工三「演算法」課程

第三次作業

**題目: K-Selection**

|  |  |  |
| --- | --- | --- |
| 班級 | ： | 資工三 |
| 姓名 | ： | 陳博琳 |
| 學號 | ： | S10859007 |

老師：陳宗禧

中華民國 110年11月10日

# 目錄

1. **簡介及問題描述……….……………..…………………………………………1**
   1. **簡介…………….…………………………………………………………………………2**
   2. **問題………………………….……………………………………………………………4**
2. **理論分析….………………………..………………………………………8**
3. **演算法則….…………………………..……………………………………10**
4. **程式設計環境架構.………………………..…………………………………12**
5. **程式.…………………………………………..………………………………14**
6. **執行結果、討論與心得.………………………..……………………………18**

參考文獻………………………………………………………….…………………22

**(一) 簡介及問題描述**

給定一 n keys 為未排序的資料(Random)，由 n 筆資料中，選第 k 小的資料，理論驗證與實驗分析該問題!

1. 簡介

從未排序的array數列中，數列中總共有n個數，尋找數列中第k小的值，觀察時間空間的變化。

2. 問題

* + 1. Insertion Sort
    2. Merge Sort
    3. Quick Sort
    4. QuickSelection
    5. Median-of-medians Select algorithm g=3,5,7

**(二) 理論分析**

1. Insertion Sort理論

將未排序的第一個element取出，插入以排序數列中的正確位子，插入後將所有比選取到的值大的element向右shift

2. Merge Sort理論

將所有element先拆成單一個，兩兩merge順便排序，直到merge成一個list

3. QuickSort理論

選取第一個element當pivot，向後插進list左邊恆小於pivot，右邊恆大於pivot。

4. QuickSelection理論

利用quicksort原理，附加隨時查看element index，如果index=k return，如果index<k 往右邊取pivot，如果index>k 往左邊取pivot

5. Median-of-medians Select algorithm

先分群，基數個數分成一群(這樣才有median)，所有的median集合再找出他們的median

**(三) 演算法則**

1. 第一個演算法(Algorithm)

Insertion Sort理論

Input: an array a[0…n-1] of n orderable element

Output: array a[0…n-1]sorted in non decreaseing order

For i<=1 to n-1 do

V<=a[i]

J<=i-1

While j>=0 and a[j]>v do

A[j+1]<=a[j]

J<=j-1

A[j+1]<=v

1. 演算法時間複雜度(time complexity)

Average case O(n^2), worstcase O(n^2)

1. 演算法空間複雜度(space complexity)

O(1), …

2. 第二個演算法(Algorithm)

MergeSort理論

Mergesort(a[0..n-1])

Input:an array a[0…n-1]by recursive mergesort

Output:array a[0…n-1]sorted in nondecreaseing order

If n>1

Copy a[0..[n/2]-1]to b [0…[n/2]-1]

Copy a[[n/2]..n-1] to c[0…[n/2]-1]

Mergesort(b [0…[n/2]-1])

Mergesort(c [0…[n/2]-1])

Merge(b,c,a)

1. 演算法時間複雜度(time complexity)

Average case O(n logn), worstcase O(nlogn)

1. 演算法空間複雜度(space complexity)

O(n),

3. 第三個演算法(Algorithm)

Quicksort理論

void swap(int \*a, int \*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int Partition(int \*arr, int front, int end){

int pivot = arr[end];

int i = front -1;

for (int j = front; j < end; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

i++;

swap(&arr[i], &arr[end]);

return i;

}

void QuickSort(int \*arr, int front, int end){

if (front < end) {

int pivot = Partition(arr, front, end);

QuickSort(arr, front, pivot - 1);

QuickSort(arr, pivot + 1, end);

}

}

void PrintArray(int \*arr, int size){

for (int i = 0; i < size; i++) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

}

int main() {

int n = 9;

int arr[] = {9, 4, 1, 6, 7, 3, 8, 2, 5};

std::cout << "original:\n";

PrintArray(arr, n);

QuickSort(arr, 0, n-1);

std::cout << "sorted:\n";

PrintArray(arr, n);

return 0;

}

1. 演算法時間複雜度(time complexity)

Average case O(nlogn), worst case O(n^2)

1. 演算法空間複雜度(space complexity)

O(logn)~O(n)

4. 第四個演算法(Algorithm)

**function** select(list, left, right, k)

**loop**

**if** left = right

**return** list[left]

pivotIndex := ... *// select pivotIndex between left and right*

pivotIndex := partition(list, left, right, pivotIndex)

**if** k = pivotIndex

**return** list[k]

**else if** k < pivotIndex

right := pivotIndex - 1

**else**

left := pivotIndex + 1

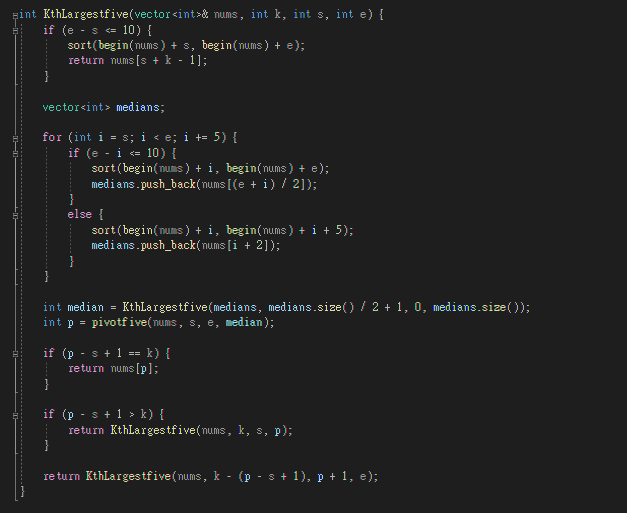
1. 演算法時間複雜度(time complexity)

Best case O(n), worstcase O(n^2)

1. 演算法空間複雜度(space complexity)

O(1),

5. 第五個演算法(Algorithm)



1. 演算法時間複雜度(time complexity)

Best caseO(n), worst case O(n)

1. 演算法空間複雜度(space complexity)

O(logn),

**(四) 程式設計環境架構**

程式設計語言、工具、環境與電腦硬體等規格說明…

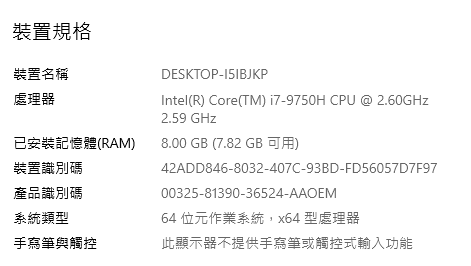
1. 程式語言

C in MS Windows

2. 程式開發工具

Visual C++ 2019(不需要system pause跟版本有關)

3. 電腦硬體

**(五) 程式 (含source code, input code, and output code)**

程式含source code, input code, and output code等…

1. 主程式

請查閱cpp檔案

2. Input Code Format

Three of examples for input use are in below….

(1)array size 6, 第5小

(2) array size 10, 第5小

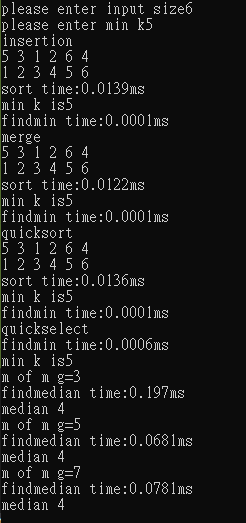
(3) array size 20, 第5小

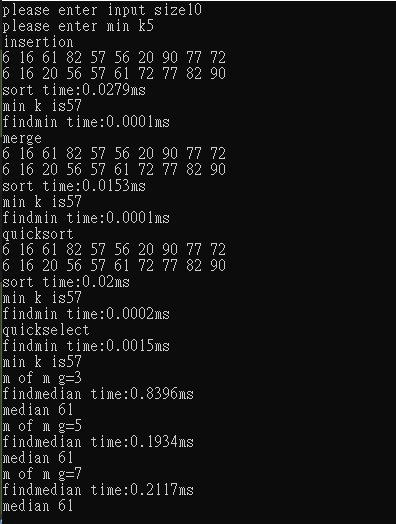
(4) array size 20, 第10小

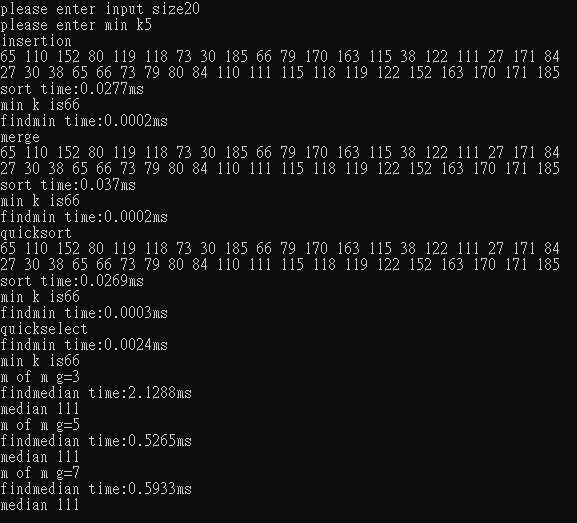
(5) array size 20, 第15小

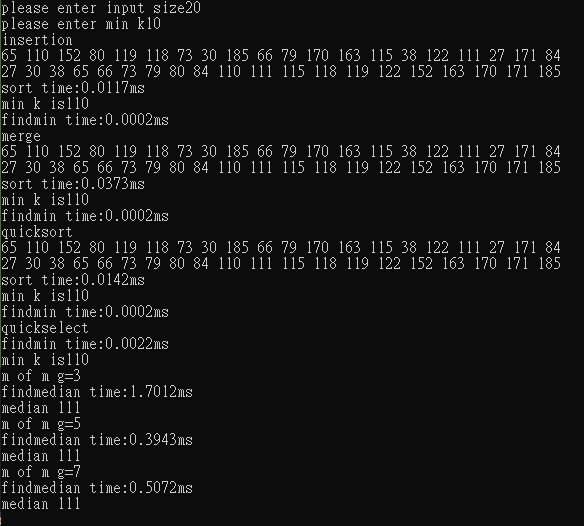
3. Output Code Format

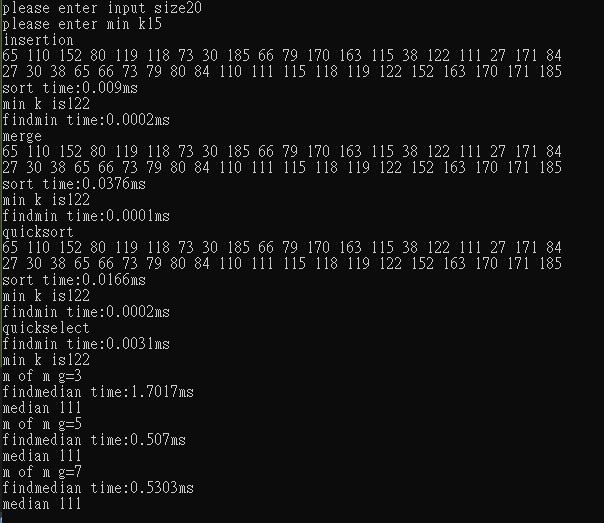
Three of examples for output use are in below….

(1) 

(2) 

(3) 

(4) 

(5) 

**(五) 執行結果、討論與心得**

執行結果與討論 (執行時間、problem *n*的大小等問題討論)等…

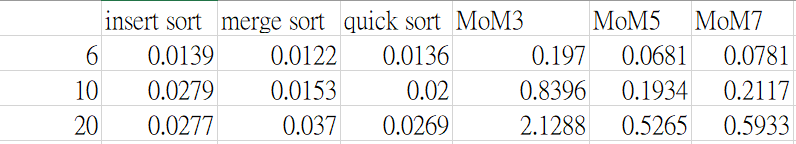
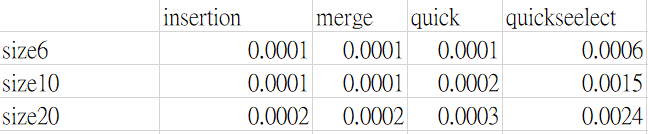
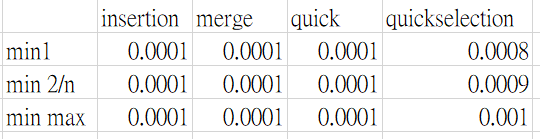
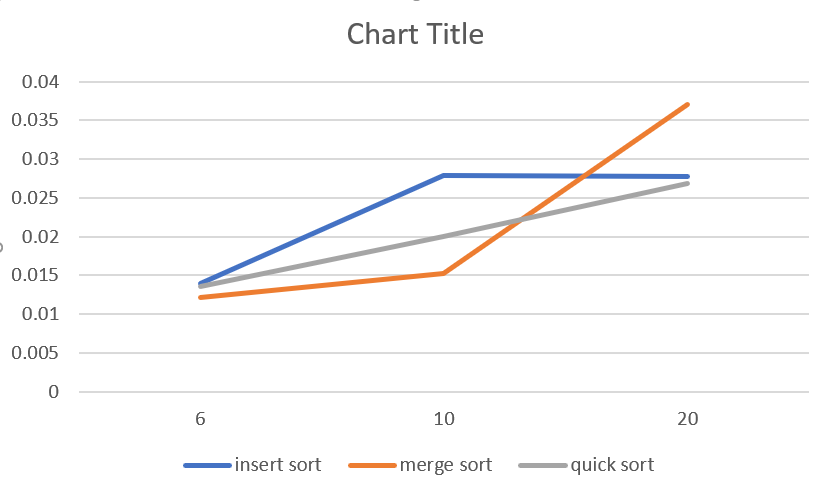
1. 執行結果

Output of program:

上方圖片已提供所有output

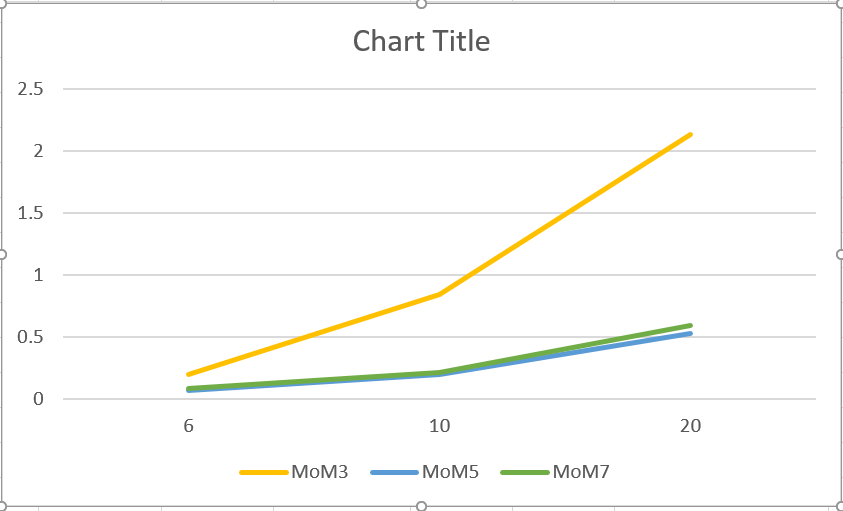
2. 討論

執行時間、問題大小等問題討論! 利用 MS Excel 畫出問題大小與執行時間的關係!

1. Running Time
2. 
3. 
4. 
5. 

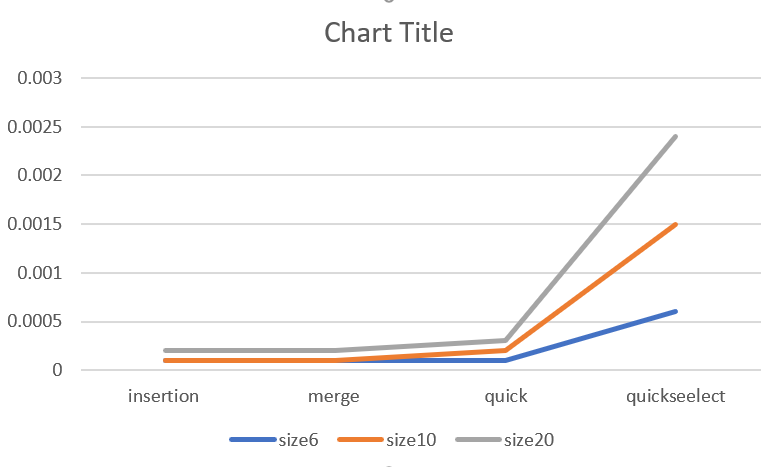
N值變化

不同array大小sorting的花費時間，這裡剛好再size20的時候取到bestcase，所以insert數據並非像size=6,10的時後線性成長

1. 

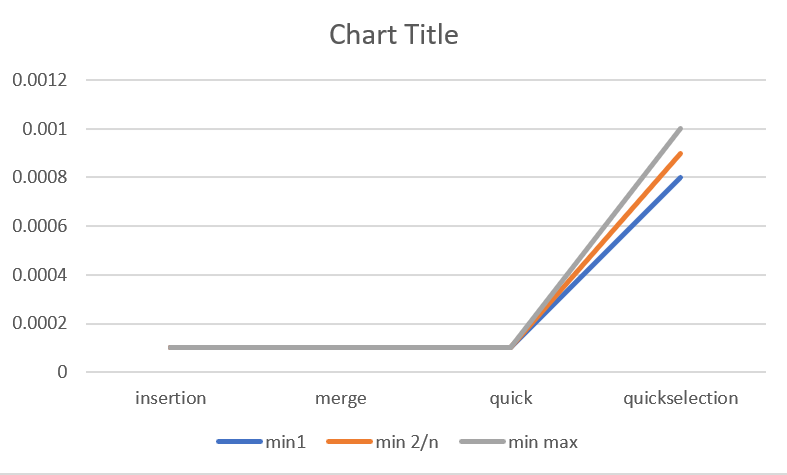
G值變化

當array size增大之後分成3個為一群的花費時間最多，因為分群後再取median的數量最多

1. 

不同n同k

同時選取第5小的情況下，quick select不論array的大小都花費最多時間

1. 

N=20情況下的k

可以發現quickselect再選取越大的數字所要花費的時間越多，其他選擇花費時間均相同

(2) Problem size *n*

執行空間的變化:

1.Insertion sort將使用1個變數裝插入的value其他用shift的方式 o(1)

2.merge sort用array存取被merge號的新list，原本的list必須被保留等再merge o(n)

3.quicksort 使用1個變數儲存povit，儲存間與選取到的povit有關o(logn)-o(n)

4.quickselect使用與sort相同空間大小povit尋找與index有關o(1)

5.median of median切分等分尋找中位數，list大小為切分的數目o(logn)

3. 心得

Array本身的排序與演算法的選擇相關，像insertsort的bestcase和worstcase差很多，可能就會倒置不同相同size但是不同排序情況下結果差很多

**參考文獻**

1. <https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E6%8E%92%E5%BA%8F%E6%B3%95%E5%85%A5%E9%96%80-%E9%81%B8%E6%93%87%E6%8E%92%E5%BA%8F%E8%88%87%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F%E6%B3%95-23d4bc7085ff>
2. https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E6%8E%92%E5%BA%8F%E6%B3%95%E9%80%B2%E9%9A%8E-%E5%90%88%E4%BD%B5%E6%8E%92%E5%BA%8F%E6%B3%95-6252651c6f7e