計算方法設計 Bonus 1

原報告 HackMD：https://hackmd.io/@liuutin9/rJ77l1pJkl

Recursive Method

Pseudo Code

```
int SelectKthSmallest(A[], l, r, kth, numsPerGroup) {
    size, numsOfGroup, excess, pivotInitPosition;
    if (numsPerGroup == 0) {
        pivotInitPosition = RandomizedSelectPivot(l, r);
    }
    else {
        size = r - l + 1;
        numsOfGroup = (size) / numsPerGroup;
        excess = (size) % numsPerGroup;
        if (excess != 0) numsOfGroup++;
        if (size <= numsPerGroup) {
            InsertionSort(A, l, r);
            pivotInitPosition = l + size / 2;
        }
        else {
            for (j = l; j <= l + numsOfGroup - 1; j++) {
                InsertionSortColumn(A, j, r, numsOfGroup);
            }
            pivotInitPosition = SelectKthSmallest(A, l + (numsPerGroup / 2) * numsOfGroup,
l + (numsPerGroup / 2 + 1) * numsOfGroup - 1, (numsOfGroup % 2 ? numsOfGroup / 2 :
numsOfGroup / 2 + 1), numsPerGroup);
        }
    }
    pivotFinalPosition = Partition(A, l, r, pivotInitPosition);
    k = pivotFinalPosition - l + 1;
    if (kth == k)
        return pivotFinalPosition;
    else if (kth < k)
        return SelectKthSmallest(A, l, pivotFinalPosition - 1, kth, numsPerGroup);
    else
        return SelectKthSmallest(A, pivotFinalPosition + 1, r, kth - k, numsPerGroup);
}

int RandomizedSelectPivot(l, r) {
    return std::rand() % (r - l + 1) + l;
}

int MedianSelectPivot(A[], l, r, numsPerGroup) {
    size = r - l + 1;
    numsOfGroup = (size) / numsPerGroup;
    excess = (size) % numsPerGroup;
    if (excess != 0) numsOfGroup++;
    if (size < numsPerGroup) {
        InsertionSort(A, l, r);
        return l + size / 2;
```

```
    }
    else {
        for (j = l; j <= l + numsOfGroup - 1; j++) {
            InsertionSortColumn(A, j, r, numsOfGroup);
        }
        return MedianSelectPivot(A, l + (numsPerGroup / 2) * numsOfGroup, l +
(numsPerGroup / 2 + 1) * numsOfGroup - 1, umsPerGroup);
    }
}

int Partition(A[], l, r, i) {
    swap(A[r], A[i]);
    pivotPosition = l;
    for (j = l; j < r; j++) {
        if (A[j] <= A[r]) swap(A[pivotPosition++], A[j]);
    }
    swap(A[pivotPosition], A[r]);
    return pivotPosition;
}

void InsertionSortColumn(int A[], int l, int r, int numsOfGroup) {
    size = (r - l + 1) / numsOfGroup;
    tmp[size] = {A[l]};
    for (int i = 1; i < size; i++) {
        tmp[i] = A[l + i * numsOfGroup];
        curr = i;
        while (tmp[curr] < tmp[curr - 1] && curr > 0) {
            swap(tmp[curr], tmp[curr - 1]);
            curr--;
        }
    }
    for (i = 0; i < size; i++) {
        A[l + i * numsOfGroup] = tmp[i];
    }
}

void InsertionSort(A[], l, r) {
    for (i = l + 1; i <= r; i++) {
        curr = i;
        while (A[curr] < A[curr - 1] && curr > l) {
            swap(A[curr], A[curr - 1]);
            curr--;
        }
    }
}
```

Time Complexity Analysis

設有 n 筆資料:

- **Group of 3:**

  T(n) = T(ceiling(n / 3)) + Θ(n) + max {T(x), T(y)}
  - **T(celing(n / 3)):** 尋找 pivot 需要花的時間
  - **Θ(n):** 做 partition 的時間
  - **max {T(x), T(y)}:** 找左半段或右半段比較久的時間

  x, y >= 2(ceiling(ceiling(n / 3) / 2) - 2) >= n / 3 - 4
  - 每個完整的 group 會有 1 個大於 pivot 的數
  - ceiling(ceiling(n / 3) / 2) 表示 pivot 那組和 median 比他大的 group 數量
  - -2 表示可能不是完整的 group 的數量

  Therefore, $n / 3 - 4 \geq n / 4 \; if \; n \geq 48$. x, y is at least n / 4 for large n respectively.

  Assume T(n) > cn.
  T(n) > c * ceiling(n / 3) + c * (n / 3 - 4) + an > cn / 3 + cn / 3 - 4c + an = 2cn / 3 - 4c + an = (2c / 3 + a)n - 4c > cn

  c < an / (n / 3 + 4) when n > 0. Assume n >= 48, then n / (n / 3 + 4) >= 2.4 > 2. So, we choose c > 2a.

  將 max {T(x), T(y)} 替換成 min {T(x), T(y)} 後的時間複雜度 > cn，因此找 worst case max {T(x), T(y)} 也會 > cn。

  Time complexity: T(n) = ω(n) != O(n)

- **Group of 5:**

  T(n) = T(ceiling(n / 5)) + Θ(n) + max {T(x), T(y)}
  - **T(celing(n / 5)):** 尋找 pivot 需要花的時間
  - **Θ(n):** 做 partition 的時間
  - **max {T(x), T(y)}:** 找左半段或右半段比較久的時間

  x, y >= 3(ceiling(ceiling(n / 5) / 2) - 2) >= 3n / 10 - 6
  - 每個完整的 group 會有 3 個大於 pivot 的數
  - ceiling(ceiling(n / 5) / 2) 表示 pivot 那組和 median 比他大的 group 數量
  - -2 表示可能不是完整的 group 的數量

Therefore, x, y <= n - 3n / 10 + 6 = 7n / 10 + 6. 7n / 10 + 6 <= 3n / 4 if n >= 140. x, y is at most 3n / 4 for large n respectively.

Assume T(n) <= cn.
T(n) <= c * ceiling(n / 5) + c * (7n / 10 + 6) + an <= cn / 5 + c + 7cn / 10 + 6c + an = 9cn / 10 + 7c + an
= cn + ((-cn) / 10 + 7c + an) <= cn

c >= 10an / (n - 70) when n > 70. Assume n >= 140, then n / (n - 70) <= 2. So, we choose c >= 20a

Time complexity: T(n) = O(n)

- **Group of 7:**

T(n) = T(ceiling(n / 7)) + Θ(n) + max {T(x), T(y)}

- **T(celing(n / 7)):** 尋找 pivot 需要花的時間
- **Θ(n):** 做 partition 的時間
- **max {T(x), T(y)}:** 找左半段或右半段比較久的時間

x, y >= 4(ceiling(ceiling(n / 7) / 2) - 2) >= 2n / 7 - 8

- 每個完整的 group 會有 4 個大於 pivot 的數
- ceiling(ceiling(n / 7) / 2) 表示 pivot 那組和 median 比他大的 group 數量
- -2 表示可能不是完整的 group 的數量

Therefore, x, y <= n - 2n / 7 + 8 = 5n / 7 + 8. 5n / 7 + 8 <= 3n / 4 if n >= 224. x, y is at most 3n / 4 for large n respectively.
Assume T(n) <= cn.
T(n) <= c * ceiling(n / 7) + c * (5n / 7 + 8) + an <= cn / 7 + c + 5cn / 7 + 8c + an = 6cn / 7 + 9c + an
= cn + ((-cn) / 7 + 9c + an) <= cn

Therefore, x, y <= n - 2n / 7 + 8 = 5n / 7 + 8. 5n / 7 + 8 <= 3n / 4 if n >= 224. x, y is at most 3n / 4 for large n respectively.

Assume T(n) <= cn.
T(n) <= c * ceiling(n / 7) + c * (5n / 7 + 8) + an <= cn / 7 + c + 5cn / 7 + 8c + an = 6cn / 7 + 9c + an
= cn + ((-cn) / 7 + 9c + an) <= cn

c >= 7an / (n - 63) when n > 63. Assume n >= 224, then n / (n - 63) < 2. So, we choose c > 14a.

Time complexity: T(n) = O(n)

- **Group of 9:**

T(n) = T(ceiling(n / 9)) + Θ(n) + max {T(x), T(y)}

- **T(celing(n / 9)):** 尋找 pivot 需要花的時間
- **Θ(n):** 做 partition 的時間
- **max {T(x), T(y)}:** 找左半段或右半段比較久的時間

x, y >= 5(ceiling(ceiling(n / 9) / 2) - 2) >= 5n / 18 - 10

- 每個完整的 group 會有 4 個大於 pivot 的數
- ceiling(ceiling(n / 9) / 2) 表示 pivot 那組和 median 比他大的 group 數量
- -2 表示可能不是完整的 group 的數量

Therefore, x, y <= n - 5n / 18 + 10 = 13n / 18 + 10. 13n / 18 + 10 <= 3n / 4 if n >= 360. x, y is at most 3n / 4 for large n respectively.

Assume T(n) <= cn.
T(n) <= c * ceiling(n / 9) + c * (13n / 18 + 10) + an <= cn / 9 + c + 13cn / 18 + 10c + an = 5cn / 6 + 11c + an = cn + ((-cn) / 6 + 11c + an) <= cn

c >= 6an / (n - 66) when n > 66. Assume n >= 360, then n / (n - 66) < 2. So, we choose c > 12a.

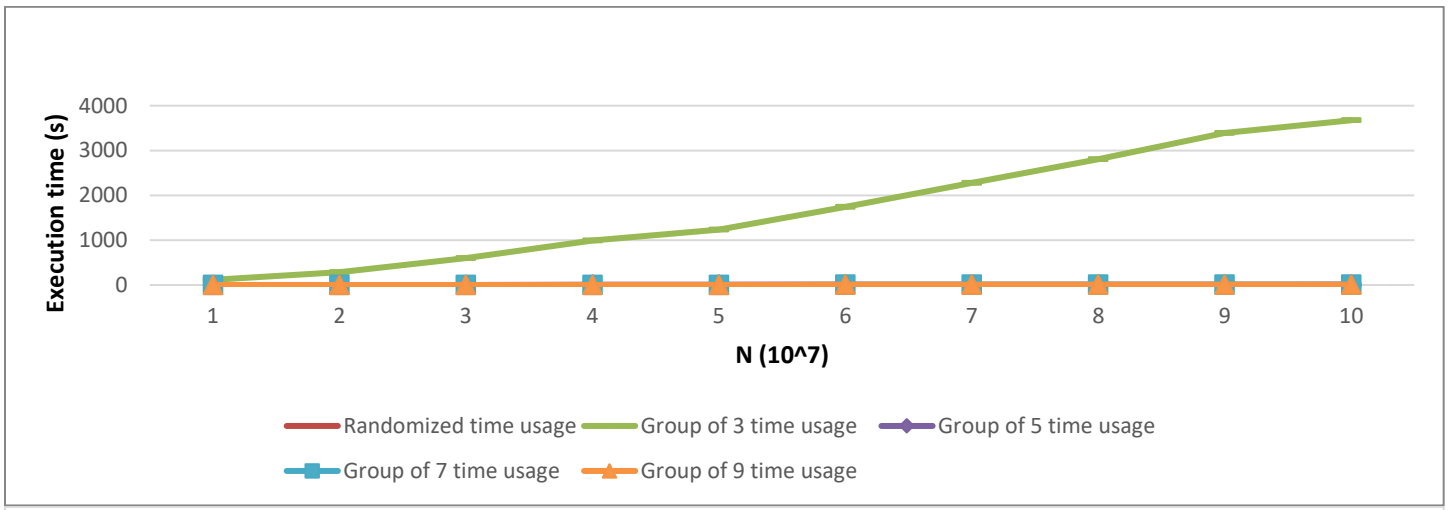
Time complexity: T(n) = O(n)

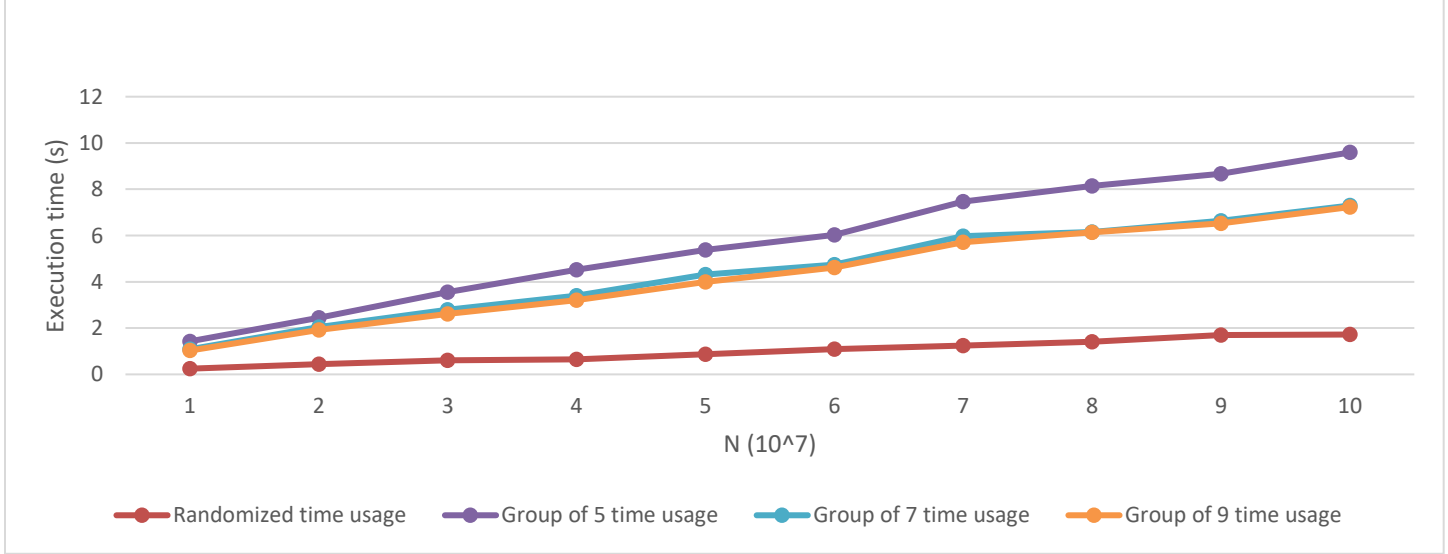
Test Result

由以下數據可以觀察到，randomized select 平均執行時間複雜度是線性的，當 group size 為 5, 7, 9 時也大致上是線性的，而 group size 為 3 時，時間複雜度隨資料量攀升的速度明顯快於其他四者。

Array size	1 * 10e7	2 * 10e7	3 * 10e7	4 * 10e7	5 * 10e7
Execution times (times)	50	50	50	50	50
Randomized time usage (s)	0.24708	0.43530	0.61172	0.64356	0.87252
Group of 3 time usage (s)	116.637	288.285	596.894	994.205	1234.79
Group of 5 time usage (s)	1.42802	2.44330	3.55082	4.51618	5.38158
Group of 7 time usage (s)	1.08526	2.05162	2.79684	3.40446	4.31722
Group of 9 time usage (s)	1.02906	1.91562	2.6124	3.20722	4.00038

Array size	6 * 10e7	7 * 10e7	8 * 10e7	9 * 10e7	1 * 10e8
Execution times (times)	50	50	50	50	50
Randomized time usage (s)	1.09072	1.24778	1.41000	1.69284	1.72172
Group of 3 time usage (s)	1743.61	2279.04	2807.06	3390.56	3675.29
Group of 5 time usage (s)	6.03228	7.46748	8.14166	8.67288	9.58924
Group of 7 time usage (s)	4.74148	5.97242	6.15724	6.63538	7.29540
Group of 9 time usage (s)	4.61310	5.70794	6.14254	6.52264	7.22966





Selective Method

Brief

把尋找 median 的 code 包裝成一個 function 後，根據觀察發現，這兩個 recursive function 其實就是一直慢慢跑到最底層就結束，因此可以透過使用 while loop 的方式實作，迴圈結束前更新 partition 的範圍，並在找到值之後直接 break 離開迴圈。

Pseudo Code

```
int SelectKthSmallest(A[], l, r, kth, numsPerGroup) {
    pivotInitPosition, pivotFinalPosition, k;
    while (true) {
        pivotInitPosition = (numsPerGroup == 0)
            ? RandomizedSelectPivot(l, r)
            : MedianSelectPivot(A, l, r, numsPerGroup);
        pivotFinalPosition = Partition(A, l, r, pivotInitPosition);
        k = pivotFinalPosition - l + 1;
        if (kth == k) {
            return A[pivotFinalPosition];
        } else if (kth < k) {
            r = pivotFinalPosition - 1;
        } else {
            l = pivotFinalPosition + 1;
            kth -= k;
        }
    }
}

int MedianSelectPivot(A[], l, r, numsPerGroup) {
    size, numsOfGroup, excess;

    while (true) {
        size = r - l + 1;
        numsOfGroup = (size) / numsPerGroup;
        excess = (size) % numsPerGroup;
        if (excess != 0) numsOfGroup++;
        if (size < numsPerGroup) {
            InsertionSort(A, l, r);
            return l + size / 2;
        }
        else {
            for (j = l; j <= l + numsOfGroup - 1; j++) {
                InsertionSortColumn(A, j, r, numsOfGroup);
            }
            r = l + (numsPerGroup / 2 + 1) * numsOfGroup - 1;
            l = l + (numsPerGroup / 2) * numsOfGroup;
        }
    }
}
```