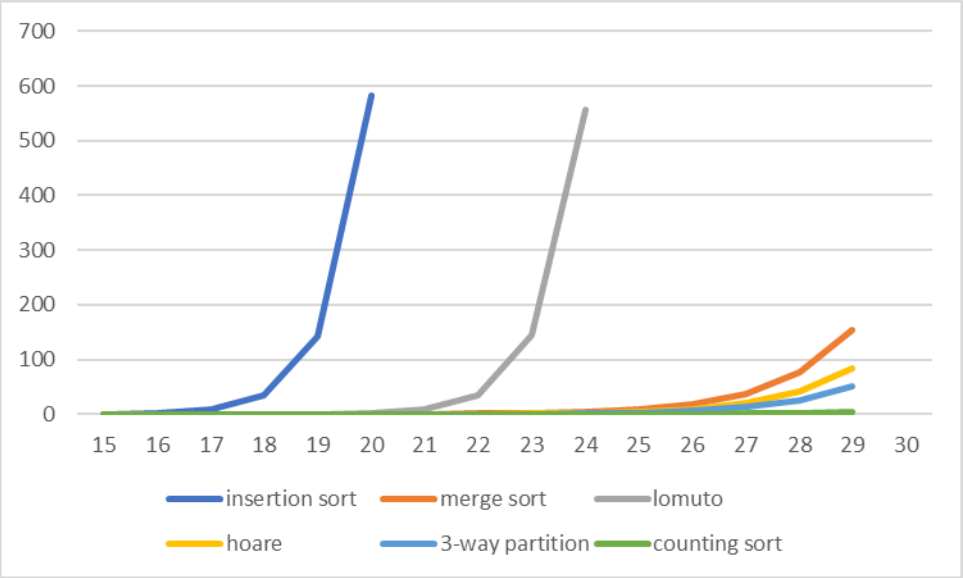
實驗數據與折線圖:



	insertion sc	merge sort	lomuto	hoare	3-way parti	counting sort
15	0.564	0.009	0.006	0.004	0.003	0
16	2.248	0.016	0.018	0.008	0.006	0.001
17	8.893	0.031	0.053	0.018	0.012	0.002
18	35.146	0.065	0.174	0.036	0.023	0.003
19	143.132	0.125	0.617	0.071	0.047	0.004
20	582.14	0.258	2.4	0.143	0.093	0.009
21		0.517	9.2	0.289	0.189	0.02
22		1.076	35.7	0.584	0.397	0.039
23		2.17	144	1.193	0.789	0.075
24		4.543	555.8	2.409	1.597	0.149
25		8.977		4.929	3.058	0.306
26		18.29		9.97	6.406	0.64
27		37.119		20.428	12.84	1.253
28		76.211		40.856	25.157	2.66
29		154.311		83.465	50.008	5.24
30						

未能實驗出的數據推測方法:

Insertion sort :k+1 的時間大約為 k 的 4 倍。

Counting sort , Merge sort , Randomized quick sort 中 hoare partition 和 3-Way partition: k+1 時間大約為 k 的兩倍。

Lomuto partition: k+1 的時間為 k 時間的 3.多倍，因此後面的數據用實驗出最後兩項的商當作後面的倍數。

	insertion sort	merge sort	lomuto	hoare	3-way parti	counting sort
15	0.564	0.009	0.006	0.004	0.003	0
16	2.248	0.016	0.018	0.008	0.006	0.001
17	8.893	0.031	0.053	0.018	0.012	0.002
18	35.146	0.065	0.174	0.036	0.023	0.003
19	143.132	0.125	0.617	0.071	0.047	0.004
20	582.14	0.258	2.4	0.143	0.093	0.009
21	2328	0.517	9.2	0.289	0.189	0.02
22	9312	1.076	35.7	0.584	0.397	0.039
23	37248	2.17	144	1.193	0.789	0.075
24	148992	4.543	555.8	2.409	1.597	0.149
25	595968	8.977	2167.62	4.929	3.058	0.306
26	2383872	18.29	8453.718	9.97	6.406	0.64
27	9535488	37.119	32969.5	20.428	12.84	1.253
28	38141952	76.211	128581	40.856	25.157	2.66
29	152567808	154.311	501466	83.465	50.008	5.24
30	610271232	308.622	1504398	166.93	100.016	10.48

實驗結果:

結果和預期比較接近的只有 counting sort 和 insertion sort，counting sort 時間複雜度和實驗結果一樣是 O(n)，insertion sort 是 O(n 平方)。

Merge sort , Randomized quick sort 的 hoare partition , 3-way partition 實驗的時間複雜度 O(n)，理論上應該要是 O(nlogn)但實驗數據 k+1 的時間除以 k 時間都非常接近 2。

Lomuto partition randomized quick sort 時間複雜度有比 O(n)大而且比 O(n 平方)小，但執行時間比另外兩種 quick sort 長很多。

程式碼來源:

https://www.geeksforgeeks.org/insertion-sort/

https://www.geeksforgeeks.org/merge-sort/

https://www.programiz.com/dsa/counting-sort

https://iq.opengenus.org/3-way-partitioning-quick-sort/

https://www.geeksforgeeks.org/quicksort-using-random-pivoting/

心得:

原本預期三種 randomized quick sort 在測時間的時候會因為隨機的關係每次實驗的誤差會很大，但可能是因為陣列數字夠大可以比較接近期望值所以每次測出來的時間都差不多。自己實作一次實驗以後才知道要做到和理論一樣的結果沒有想像中簡單，結果也和預期的有差異而且找不出來造成這些問題的原因在哪裡。

實驗程式碼:

Insertion sort:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
// Function to sort an array using
// insertion sort
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key, to one
        // position ahead of their
        // current position
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// A utility function to print an array
// of size n
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    //cout<<b;
    srand(100);
    int* arr=new int[b];
    for(int k=0;k<b;++k)
    {
        *(arr+k)=rand()%1000;
    }

    int N = b;

    double start,end;
```

```
    start=clock();
    insertionSort(arr, N);
    end=clock();
    //printArray(arr, N);
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
    delete[] arr;
    return 0;
}
```

Merge sort:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid,
           int const right)
{
    int const subArrayOne = mid - left + 1;
    int const subArrayTwo = right - mid;

    // Create temp arrays
    int *leftArray = new int[subArrayOne],
        *rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (int i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (int j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    int indexOfSubArrayOne
        = 0, // Initial index of first sub-array
        indexOfSubArrayTwo
        = 0; // Initial index of second sub-array
    int indexOfMergedArray
        = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne
           && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }
```

```
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray]
            = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // right[], if there are any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
    delete[] leftArray;
    delete[] rightArray;
}

// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return; // Returns recursively

    int mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

// UTILITY FUNCTIONS
// Function to print an array
void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
}


int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    //cout<<b;
    srand(100);
    int* arr=new int[b];
    for(int k=0;k<b;++k)
    {
        *(arr+k)=rand()%1000;
    }
```

```
int N = b;

double start,end;

start=clock();
mergeSort(arr, 0, N-1);
end=clock();
//printArray(arr, N);
cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;

//printArray(arr, N);

delete[] arr;

return 0;
```

Counting sort:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

void countSort(int array[], int size) {
  // The size of count must be at least the (max+1) but
  // we cannot assign declare it as int count(max+1) in C++ as
  // it does not support dynamic memory allocation.
  // So, its size is provided statically.
  int *output=new int[size];
  int *count=new int[size];
  int max = array[0];

  // Find the largest element of the array
  for (int i = 1; i < size; i++) {
    if (array[i] > max)
      max = array[i];
  }

  // Initialize count array with all zeros.
  for (int i = 0; i <= max; ++i) {
    count[i] = 0;
  }

  // Store the count of each element
  for (int i = 0; i < size; i++) {
    count[array[i]]++;
  }

  // Store the cummulative count of each array
  for (int i = 1; i <= max; i++) {
    count[i] += count[i - 1];
  }

  // Find the index of each element of the original array in count array, and
  // place the elements in output array
  for (int i = size - 1; i >= 0; i--) {
    output[count[array[i]] - 1] = array[i];
    count[array[i]]--;
  }
```

```
  // Copy the sorted elements into original array
  for (int i = 0; i < size; i++) {
    array[i] = output[i];
  }
}

// Function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++)
    cout << array[i] << " ";
  cout << endl;
}

// Driver code
int main() {


  int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    //cout<<b;
    srand(100);
    int* array=new int[b];
    for(int k=0;k<b;++k)
    {
        *(array+k)=rand()%1000;
    }

    int N = b;

    double start,end;

    start=clock();
    countSort(array,b);
    end=clock();
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
    //printArray(array,b);
    delete[] array;
    return 0;
}
```

randomized quick sort (Lomuto partition:

```
#include <iostream>
using namespace std;
int partition(int arr[], int low, int high)
{
    // pivot
    int pivot = arr[high];

    // Index of smaller element
    int i = (low - 1);
```

```
    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller
        // than or equal to pivot
        if (arr[j] <= pivot) {

            // increment index of
            // smaller element
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[high]);

    return partition(arr, low, high);
}
void quickSort(int arr[], int low, int high)
{
    if (low < high) {

        /* pi is partitioning index,
        arr[p] is now
        at right place */
        int pi = partition_r(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    //cout<<b;
    srand(100);
```

```
    int* arr=new int[b];
    for(int k=0;k<b;++k)
    {
        *(arr+k)=rand()%1000;
    }

    int N = b;

    double start,end;

    start=clock();
    quickSort(arr, 0, N-1);
    end=clock();
    //printArray(arr, N);
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;

    //printArray(arr, N);

    delete[] arr;

    return 0;
}
```

randomized quick sort (Hoare partition:

```
#include <cstdlib>
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int partition(int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low - 1, j = high + 1;

    while (true) {

        // Find leftmost element greater than
        // or equal to pivot
        do {
            i++;
        } while (arr[i] < pivot);

        // Find rightmost element smaller than
        // or equal to pivot
        do {
            j--;
        } while (arr[j] > pivot);

        // If two pointers met
        if (i >= j)
            return j;

        swap(arr[i], arr[j]);
    }
}
```

```
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[low]);

    return partition(arr, low, high);
}

// The main function that implements QuickSort
// arr[] --> Array to be sorted,
// low  --> Starting index,
// high  --> Ending index
void quickSort(int arr[], int low, int high)
{
    if (low < high) {
        // pi is partitioning index,
        // arr[p] is now at right place
        int pi = partition_r(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

// Function to print an array
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {b=b*2;}
    srand(100);
    int* arr=new int[b];
    for(int k=0;k<b;++k)
    {*(arr+k)=rand()%1000;}
    int N = b;
    double start,end;
    start=clock();
    quickSort(arr, 0, N-1);
    end=clock();
    //printArray(arr, N);
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
    //printArray(arr, N);
    delete[] arr;
    return 0;
```

randomized quick sort (3-way partition:

```
#include<iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

void TWP(int arr[], int m, int n, int &begin, int &end)
{
    srand(time(NULL));
    int random=rand()%(n-m)+m;
  int pivot = arr[random];
  begin = m, end = n;

  for(int i = begin + 1 ; i <= end; i++)
  {
    if(arr[i] < pivot)
    {
      swap(arr[i], arr[begin]);
      begin++;
    }
    else if(arr[i] > pivot)
    {
      swap(arr[i], arr[end]);
      end--;
      i--;
    }
  }
}

void quicksort(int *arr, int m, int n)
{
  if(m>=n)
  {
    return;
  }
  int begin, end;
  TWP(arr, m, n, begin, end);
  quicksort(arr, m, --begin);
  quicksort(arr, ++end, n);
}

int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    srand(100);
    int* arr=new int[b];
    for(int k=0;k<b;++k)
    {
        *(arr+k)=rand()%1000;
    }
    int N = b;
    double start,end;
```

```
start=clock();
quicksort(arr, 0, b-1);
end=clock();
//printArray(arr, N);
cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;

//printArray(arr, N);

delete[] arr;

return 0;
```