**Bài 1:**

#include "stdio.h"

#include "time.h"

#include "stdlib.h"

void initializeArray(int arr[], int n, int min, int max) {

    for (int i = 0; i < n; i++) {

        arr[i] = min +(rand() % (max - min + 1));

    }

}

int partition(int a[], int  left, int right,int \*shiftLeft,int \*shiftRight){

    int pivot = a[right];

    right--;

    while(left<right){

        while(a[left]<pivot){

            left++;

            (\*shiftLeft)++;

        }

        while(a[right]>pivot){

            right--;

            (\*shiftRight)++;

        }

        if(left<right){

            int temp=a[right];

            a[right]=a[left];

            a[left]=temp;

        }

    }

    int temp=pivot;

    pivot=a[left];

    a[left]=temp;

    return left;

}

void quickSort(int a[], int left, int right,int \*shiftLeft,int \*shiftRight){

    if (left >= right) return;

    int indexPivot = partition(a, left, right,shiftLeft,shiftRight);

    quickSort(a,left,indexPivot - 1,shiftLeft,shiftRight);

    quickSort(a,indexPivot + 1, right,shiftLeft,shiftRight);

}

int main(){

    time\_t t;

    srand((unsigned)time(&t));

    int arr[10];

    int k[4]={10,100,1000,10000};

    for(int i=0;i<4;i++){

        int totalShiftLeft=0,totalShiftRight=0;

        printf("k=%d\n",k[i]);

        for(int j=0;j<k[i];j++){

            int shiftLeft=0,shiftRight=0;

            initializeArray(arr,10,-100,100);

            quickSort(arr,0,9,&shiftLeft,&shiftRight);

            totalShiftLeft+=shiftLeft;

            totalShiftRight+=shiftRight;

        }

        printf("Average shift left: %0.2f\n",(float)totalShiftLeft/k[i]);

        printf("Average shift right: %0.2f\n",(float)totalShiftRight/k[i]);

    }

    return 0;

}

**Bài 2:**

**1. Bubble Sort**

* **Đặc điểm:** So sánh từng cặp phần tử liền kề và hoán đổi nếu cần thiết. Tiếp tục quét qua mảng nhiều lần cho đến khi mảng được sắp xếp.
* **Độ phức tạp thời gian:**
  + **Tốt nhất:** O(n)O(n)O(n) (mảng đã được sắp xếp trước).
  + **Tệ nhất và trung bình:** O(n2)O(n^2)O(n2).

| **Trường hợp** | **Số bước thực hiện (gần đúng)** |
| --- | --- |
| n=5n = 5n=5: 3 2 9 7 5 | ~10 so sánh + 7 hoán đổi = 17 bước |
| n=10n = 10n=10: 10 9 8 ... 1 | ~45 so sánh + 45 hoán đổi = 90 bước |
| n=20n = 20n=20: 4 3 2 ... | ~190 so sánh + 110 hoán đổi = 300 bước |
| n=20n = 20n=20: 1 2 3 ... | ~19 so sánh + 0 hoán đổi = 19 bước |

**2. Selection Sort**

* **Đặc điểm:** Tìm phần tử nhỏ nhất và đặt vào vị trí đầu tiên, lặp lại cho phần còn lại của mảng.
* **Độ phức tạp thời gian:** Luôn là O(n2)O(n^2)O(n2) (bất kể dữ liệu).

| **Trường hợp** | **Số bước thực hiện (gần đúng)** |
| --- | --- |
| n=5n = 5n=5: 3 2 9 7 5 | ~10 so sánh + 4 hoán đổi = 14 bước |
| n=10n = 10n=10: 10 9 8 ... 1 | ~45 so sánh + 9 hoán đổi = 54 bước |
| n=20n = 20n=20: 4 3 2 ... | ~190 so sánh + 19 hoán đổi = 209 bước |
| n=20n = 20n=20: 1 2 3 ... | ~190 so sánh + 19 hoán đổi = 209 bước |

**3. Insertion Sort**

* **Đặc điểm:** Chèn từng phần tử vào đúng vị trí của nó trong danh sách đã sắp xếp trước đó.
* **Độ phức tạp thời gian:**
  + **Tốt nhất:** O(n)O(n)O(n) (mảng đã được sắp xếp trước).
  + **Tệ nhất và trung bình:** O(n2)O(n^2)O(n2).

| **Trường hợp** | **Số bước thực hiện (gần đúng)** |
| --- | --- |
| n=5n = 5n=5: 3 2 9 7 5 | ~7 so sánh + 5 hoán đổi = 12 bước |
| n=10n = 10n=10: 10 9 8 ... 1 | ~45 so sánh + 45 hoán đổi = 90 bước |
| n=20n = 20n=20: 4 3 2 ... | ~190 so sánh + 110 hoán đổi = 300 bước |
| n=20n = 20n=20: 1 2 3 ... | ~19 so sánh + 0 hoán đổi = 19 bước |

**4. Quick Sort**

* **Đặc điểm:** Chọn một phần tử làm "pivot," chia mảng thành hai phần (nhỏ hơn và lớn hơn pivot), rồi tiếp tục sắp xếp các phần.
* **Độ phức tạp thời gian:**
  + **Tốt nhất và trung bình:** O(nlog⁡n)O(n \log n)O(nlogn).
  + **Tệ nhất:** O(n2)O(n^2)O(n2) (mảng đã sắp xếp ngược).

| **Trường hợp** | **Số bước thực hiện (gần đúng)** |
| --- | --- |
| n=5n = 5n=5: 3 2 9 7 5 | ~7 so sánh + 4 chia = 11 bước |
| n=10n = 10n=10: 10 9 8 ... 1 | ~30 so sánh + 10 chia = 40 bước |
| n=20n = 20n=20: 4 3 2 ... | ~85 so sánh + 20 chia = 105 bước |
| n=20n = 20n=20: 1 2 3 ... | ~85 so sánh + 20 chia = 105 bước |

**Kết luận:**

* **Dữ liệu ngẫu nhiên hoặc giảm dần:**
  + **Quick Sort** là hiệu quả nhất với O(nlog⁡n)O(n \log n)O(nlogn).
  + **Bubble Sort, Selection Sort, và Insertion Sort** mất nhiều bước hơn (tương tự O(n2)O(n^2)O(n2)).
* **Dữ liệu đã sắp xếp:**
  + **Insertion Sort** là nhanh nhất (O(n)O(n)O(n)).
  + **Bubble Sort** cũng nhanh khi được tối ưu hóa (O(n)O(n)O(n)).
  + **Selection Sort** và **Quick Sort** vẫn thực hiện nhiều bước.

Chọn thuật toán phù hợp tùy theo đặc tính dữ liệu đầu vào!