算法设计与分析_第2次作业

各种排序算法的实现与分析

请用任意语言（C/C++/Java/Python等），实现选择排序、插入排序、快速排序、归并排序。

选择排序:

```c
#include "stdio.h"
#include "time.h"

void sort (int nums[], int n){
    if(nums == NULL || n <= 0){
        return ;
    }
    int i, j, t;
    clock_t st = clock();
    for(i = 0; i < n; i++){
        int min = nums[i];
        for(j = i; j < n; j++){
            if(min > nums[j]){
                t = nums[j];
                nums[j] = min;
                min = t;
            }
        }
        nums[i] = min;
    }
    clock_t ft = clock();
    double dt = (ft - st) * 1.0 / CLOCKS_PER_SEC;

    for(i = 0; i < n ; i++){
        printf("%4d ", nums[i]);
    }

    printf("\n duration: %f s\n", dt);
}

int main(void) {
    int n, nums[1000005];
    scanf("%d", &n);
    int i;
    for(i = 0; i < n; i++){
        scanf("%d", &nums[i]);
    }
    sort(nums, n);


    return 0;
}
```

插入排序:

```c
#include "stdio.h"
#include <time.h>

void sort(int nums[], int n){
    int i, j, cur;
    clock_t st = clock();
    for(i = 1; i < n; i++){
        cur = nums[i];
        for(j = i - 1; j >= 0 && nums[j] > cur; j--){
            nums[j + 1] = nums[j];
        }
        nums[j + 1] = cur;
    }
    clock_t ft = clock();
    double dt = (ft - st) * 1.0 / CLOCKS_PER_SEC;
    for(i = 0; i < n; i++){
        printf("%4d ", nums[i]);
    }
    printf("\n duration: %f s\n", dt);
}

int main(void){
    int n, nums[1000005];
    scanf("%d", &n);
    int i;
    for(i = 0; i < n; i++){
        scanf("%d", &nums[i]);
    }
    sort(nums, n);
    return 0;
}
```

快速排序:

```c
#include "stdio.h"
#include "time.h"

int n, nums[1000005];

int keySort(int nums[], int L, int H){
    int key, count = 1, temp;
    int M = L + (H - L) / 2;
    if (nums[L] > nums[H]){
        temp = nums[L];
        nums[L] = nums[H];
        nums[H] = temp;
    }
    if (nums[M] > nums[H]){
        temp = nums[H];
        nums[H] = nums[M];
        nums[M] = temp;
    }
    if (nums[L] < nums[M]){
        temp = nums[L];
        nums[L] = nums[M];
        nums[M] = temp;
    }
    nums[0] = nums[L];
    key = nums[L];
    while (L < H){
        while (L < H && nums[H] >= key)
            --H;
        nums[L] = nums[H];
        while (L < H && nums[L] <= key)
            ++L;
        nums[H] = nums[L];
        count++;
    }
    nums[L] = nums[0];
    return L;
}

void Sort(int nums[], int L, int H){
    int k;
    if (L < H){
        k = keySort(nums, L, H);
        Sort(nums, L, k - 1);
        Sort(nums, k + 1, H);
    }
}

int main(void){
    int i;
    scanf("%d", &n);
    for (i = 1; i <= n; i++){
        scanf("%d", &nums[i]);
    }
    clock_t st = clock();
    Sort(nums, 1, n);
    clock_t ft = clock();
    double dt = (ft - st) * 1.0 / CLOCKS_PER_SEC;
    printf("\n duration: %f s\n", dt);
    return 0;
}
```

归并排序:

```
#include "stdio.h"
#include "time.h"

int res[1000005], nums[1000005];

void merge(int L, int M, int H){
    int i = L, j = M + 1, k = L, n;
    for(n = L; n <= H; n++){
        res[n] = nums[n];
    }
    while(i <= M && j <= H){
        if(res[i] > res[j]){
            nums[k] = res[j];
            j++;
        }else{
            nums[k] = res[i];
            i++;
        }
        k++;
    }
    if(i <= M){
        for(n = i; n <= M; n++){
            nums[k++] = res[n];
        }
    }
}

void mergeSort(int L, int H){
    if(L + 1 >= H){
        if(nums[L] > nums[H]){
            int t = nums[L];
            nums[L] = nums[H];
            nums[H] = t;
        }
    }else{
        int M = (L + H)  / 2;
        mergeSort(L, M);
        mergeSort(M + 1, H);
        merge(L, M, H);
    }
}

int main(void){
    int n;
    scanf("%d", &n);
    int i;
    for(i = 0; i < n; i++){
        scanf("%d", &nums[i]);
    }
    clock_t st = clock();
    mergeSort(0, n - 1);
    clock_t ft = clock();
    double dt = (ft - st) * 1.0 / CLOCKS_PER_SEC;
    for(i = 0; i < n; i++){
        printf("%4d ", nums[i]);
    }
    printf("\n duration: %f s\n", dt);

    return 0;
}
```

问题：

1. 生成不同规模、不同分布的测试数据，对比测试上述4种代码的运行表现。

   （要求数据规模至少包括1万、10万、100万，分布要求必须包含一定范围内的随机数据。下表仅写排序运行时间，不要包括读入数据的时间。）

答：（至少包含以下内容，推荐对是否有重复元素附加测试）

	选择排序	插入排序	快速排序	归并排序
1万正序	0.151000 s	0.000000 s	0.000000 s	0.001000 s
1万逆序	0.195000 s	0.163000 s	0.001000 s	0.001000 s
1万随机	0.342000 s	0.081000 s	0.002000 s	0.002000 s
10万正序	14.904000 s	0.001000 s	0.005000 s	0.006000 s
10万逆序	20.961000 s	19.095000 s	0.007000 s	0.008000 s
10万随机	28.099000 s	9.745000 s	0.015000 s	0.019000 s
100万正序		0.000000 s	0.056000 s	0.068000 s
100万逆序	2920.444000 s	1204.200000 s	0.076000 s	0.091000 s
100万随机	2677.662000 s	634.461000 s	0.187000 s	0.172000 s

附：各规模随机数的说明

1万随机：（应至少包括范围，是否有重复元素，分布：均匀分布即可）

10万随机：（应至少包括范围，是否有重复元素，分布：均匀分布即可）

100万随机：（应s至少包括范围，是否有重复元素，分布：均匀分布即可）

生成随机数据的方法：（以10万规模随机数为例）

```python
import random

n = int(input())
print(n)
LR = min(10000, n)
for _ in range(n):
    print(random.randint(-LR, LR), end=" ")
```

2.实际的排序问题输入，元素可能有重复，如果输入中有较多重复元素，如何改进快速排序，使划分效率更高？请写出改进后的核心代码或算法

答：优化找数，优化重复

```c
#include "stdio.h"
#include "time.h"

int n, nums[1000005];

int keySort(int nums[], int L, int H){
    int key, count = 1, temp;
    int M = L + (H - L) / 2;
    if (nums[L] > nums[H]){
        temp = nums[L];
        nums[L] = nums[H];
        nums[H] = temp;
    }
    if (nums[M] > nums[H]){
        temp = nums[H];
        nums[H] = nums[M];
        nums[M] = temp;
    }
    if (nums[L] < nums[M]){
        temp = nums[L];
        nums[L] = nums[M];
        nums[M] = temp;
    }
    nums[0] = nums[L];
    key = nums[L];
    while (L < H){
        while (L < H && nums[H] >= key)
            --H;
        nums[L] = nums[H];
        while (L < H && nums[L] <= key)
            ++L;
        nums[H] = nums[L];
        count++;
    }
    nums[L] = nums[0];
    return L;
}

void Sort(int nums[], int L, int H){
    int k;
    if (L < H){
        k = keySort(nums, L, H);
        Sort(nums, L, k - 1);
        Sort(nums, k + 1, H);
    }
}

int main(void){
    int i;
    scanf("%d", &n);
    for (i = 1; i <= n; i++){
        scanf("%d", &nums[i]);
    }
    clock_t st = clock();
    Sort(nums, 1, n);
    clock_t ft = clock();
    double dt = (ft - st) * 1.0 / CLOCKS_PER_SEC;
    printf("\n duration: %f s\n", dt);
    return 0;
}
```