实验一 排序算法 实验报告

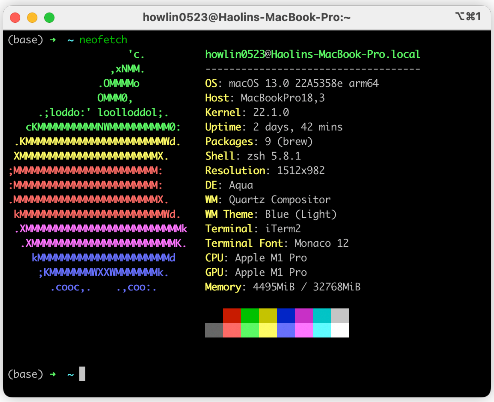
PB18061443 江昊霖

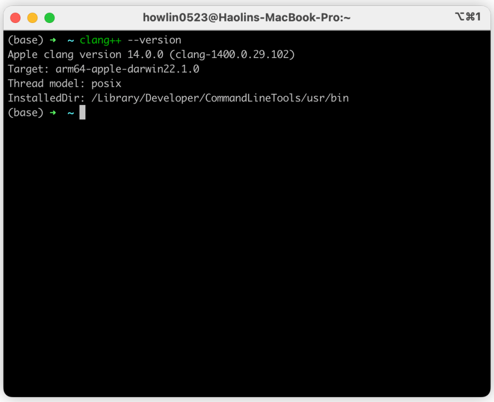
2022 年 10 月 11 日

1 实验内容

1. 排序 n 个元素，元素为随机生成的 0 到 $2^{15}-1$ 之间的整数，n 的取值为：$2^3, 2^6, 2^9, 2^{12}, 2^{15}, 2^{18}$。
2. 实现以下算法：堆排序，快速排序，归并排序，计数排序。

2 实验设备和环境





3 实验方法和步骤

3.1 堆排序

```
void heapify(vector<int> &arr, int N, int i)
{
    int largest = i;

    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])
        largest = left;

    if (right < N && arr[right] > arr[largest])
        largest = right;

    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}

void heapSort(vector<int> &arr, int N)
{
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    for (int i = N - 1; i >= 0; i--)
    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
```

```
    }
}
```

3.2 快速排序

```
int partition(vector<int> &arr, int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {

        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(vector<int> &arr, int low, int high)
{
    if (low < high)
    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
```

```
    }
}
```

3.3 归并排序

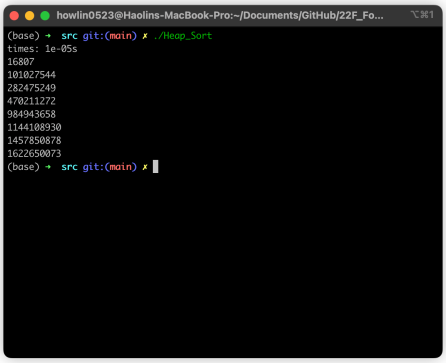
```
void merge(vector<int> &arr, int front, int mid, int end)
{
    vector<int> LeftSubarr(arr.begin() + front, arr.begin() + mid + 1);
    vector<int> RightSubarr(arr.begin() + mid + 1, arr.begin() + end + 1);
    int idxLeft = 0, idxRight = 0;
    LeftSubarr.insert(LeftSubarr.end(), numeric_limits<int>::max());
    RightSubarr.insert(RightSubarr.end(), numeric_limits<int>::max());
    for (int i = front; i <= end; i++)
    {
        if (LeftSubarr[idxLeft] < RightSubarr[idxRight])
        {
            arr[i] = LeftSubarr[idxLeft];
            idxLeft++;
        }
        else
        {
            arr[i] = RightSubarr[idxRight];
            idxRight++;
        }
    }
}

void mergeSort(vector<int> &arr, int front, int end)
{
    if (front >= end)
        return;
    int mid = (front + end) / 2;
```

```
    mergeSort(arr, front, mid);
    mergeSort(arr, mid + 1, end);
    merge(arr, front, mid, end);
}
```

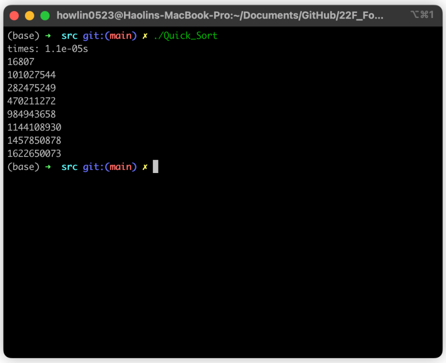
3.4 计数排序

```
void countingSort(vector<int> &arr)
{
    int max = *max_element(arr.begin(), arr.end());
    int min = *min_element(arr.begin(), arr.end());
    int range = max - min + 1;

    vector<int> count(range), output(arr.size());
    for (int i = 0; i < arr.size(); i++)
        count[arr[i] - min]++;

    for (int i = 1; i < count.size(); i++)
        count[i] += count[i - 1];

    for (int i = arr.size() - 1; i >= 0; i--)
    {
        output[count[arr[i] - min] - 1] = arr[i];
        count[arr[i] - min]--;
    }

    for (int i = 0; i < arr.size(); i++)
        arr[i] = output[i];
}
```

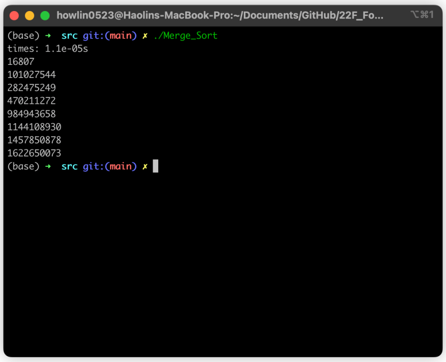
4 实验结果与分析



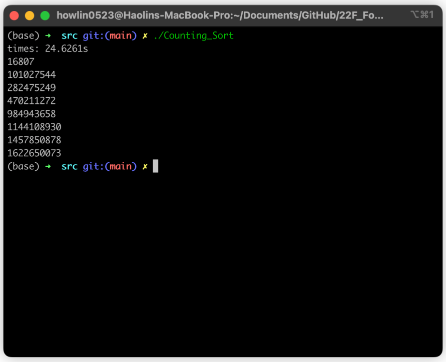
(a) 堆排序



(b) 快速排序

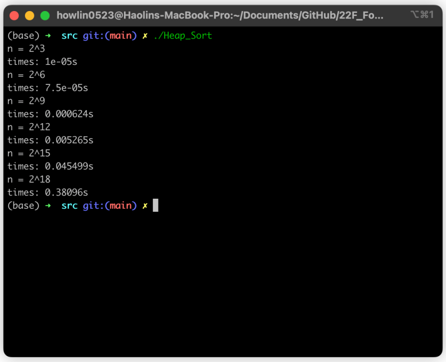


(c) 归并排序

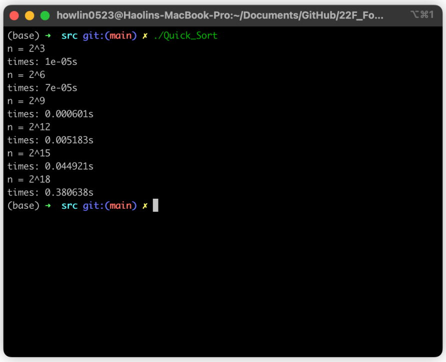


(d) 计数排序

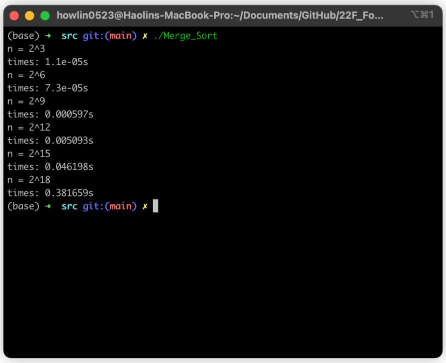
图 1: $n = 2^3$ 时排序结果



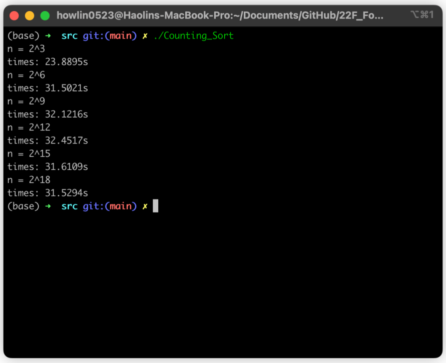
(a) 堆排序



(b) 快速排序

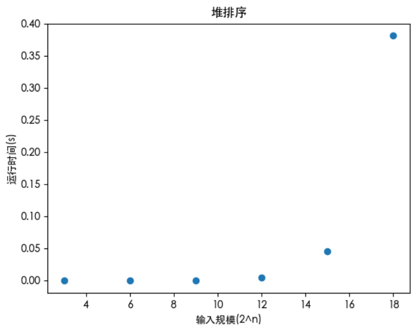


(c) 归并排序

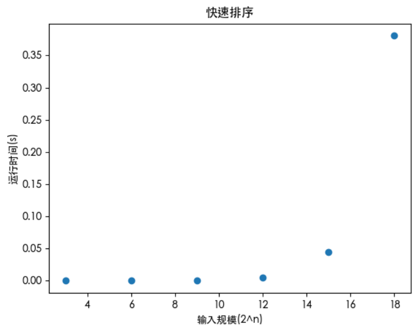


(d) 计数排序

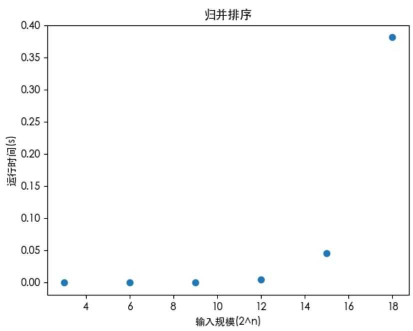
图 2: 六个输入规模运行时间



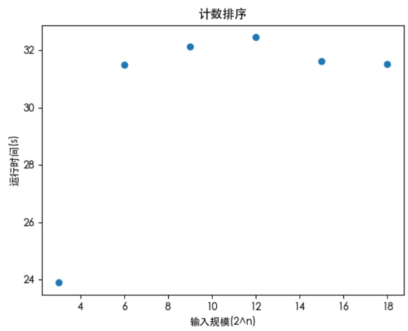
(a) 堆排序



(b) 快速排序



(c) 归并排序



(d) 计数排序

图 3: 六个输入规模运行时间

结果分析

堆排序、快速排序、归并排序的运行时间符合理论时间复杂度 $O(n \log n)$, 而计数排序符合 $O(n+k)$, 计数排序花的时间相对久是因为需要遍历数组得到 max 和 min

堆排序、快速排序、归并排序三个排序算法在 6 个输入规模下所花的时间差不多，计数排序由于输入的随机数大小 $k \in (0, 2^{15}-1)$, 时间复杂度比前三个都要高出不少