1 测试程序的设计思路

根据评分标准中要求，我们首先对 HeapSort.h 进行设计。我把实现堆排序分为两步，首先是将序列转化为最大数为根节点，所有子节点比父节点小的形式。为此我设计了调整堆（表现形式为数组）的函数 adjustHeap，该函数通过递归的方式调整数组中元素的位置，将二叉树调整为上述最大数为根节点的形式。在调整结束之后，即可用 Heapsort 函数对该堆进行排序。通过循环不断地将此堆中的最大值取出放在数组的末尾，在通过上述 adjustHeap 函数调整新的堆，即可得到一个升序的序列，排序完成。

```cpp
template <typename t>
void adjustHeap (std::vector<t>& arr, int end, int i)
{
    int parent = i;
    int max = parent;
    int l_child = 2 * i + 1;
    int r_child = 2 * i + 2;

    if (l_child <= end && arr[l_child] > arr[max])
        max = l_child;
    if (r_child <= end && arr[max] < arr[r_child])
        max = r_child;

    if (max != parent) {
        swap(arr[max], arr[parent]);
        adjustHeap (arr, end, max);
    }

}

template <typename t>
void Heapsort (std::vector<t>& arr, int len) {
    for (int i = len / 2 - 1; i >= 0; --i)
        adjustHeap(arr, len - 1, i);

    for (int n = len - 1; n > 0; n --){
        swap (arr[0], arr[n]);

        adjustHeap(arr, n - 1, 0);
    }
}
```

图 1: function in HeapSort.h

随后我们设计 test.cpp 文件中的函数。

关于检测排序正确性的 check 函数比较简单。因为我们最终生成一个升序序列，所以只循环比较每一组前后元素的大小即可验证最终排序是否为升序序列。若前一个元素更大，则证明排序错误

```cpp
template <typename t>
bool check(vector<t>& arr, int len) {
    for (int i = 1; i < len; i ++) {
        if (arr[i] < arr[i - 1]) {
            return false;
        }
    }
    return true;
}
```

图 2: function check

除此之外我设计了一个生成随机序列的函数 RandomQ，其中第一个序列 vAllIndices 是存放随机数值的序列，而第二个序列 vAvaibleIndices 则是存放索引的序列。通过随机数取模的 hash 手段来构建一个简单的随机序列，为了尽量保证不碰撞需要设计随机数值大于索引。

```cpp
vector<int> RandomQ (int n, int N) {
    srand((int)time(0));
    vector<int> vAllIndices(N);
    vector<int> vAvailableIndices(n);
    for(size_t i = 0; i < vAllIndices.size(); i++)
    {
        vAllIndices[i]=i;
    }
    for(size_t i = 0; i < vAvailableIndices.size(); i++)
    {
        int idx = random() % vAllIndices.size();
        vAvailableIndices[i] = vAllIndices[idx];
    }
    return vAvailableIndices;
}
```

图 3: function RandomQ

最后则是主函数的设计。首先是对变量进行初始化，重复序列随机数值设定为 0-1000，随机序列随机数值设定为 0-100000000。之后的测试中首先都是生成序列，然后再深度拷贝出另一个序列以分别采用 sort_heap 函数和 Heapsort 函数进行排序。同时利用 chrono 进行计时。最后根据 check 函数的判断结果输出测试是否通过。正序逆序序列就用累加累减生成。

```cpp
int main() {
    int size = 1000000;
    int numsize = 100000000;
    int numsize1 = 1000;

    vector<int> Random = RandomQ(size, numsize);
    vector<int> Random_c = Random;
    cout << "testing the random\n";

    auto start1 = chrono::high_resolution_clock::now();
    make_heap(Random.begin(), Random.end());
    sort_heap(Random.begin(), Random.end());
    auto end1 = chrono::high_resolution_clock::now();
    auto duration1 = chrono::duration_cast<chrono::microseconds>(end1 - start1);
    cout << "sort_heap:" << duration1.count() << " \n";
    cout << "Sorting Correctness: " << (check(Random, size) ? "Passed" : "Failed") << "\n";

    auto start2 = chrono::high_resolution_clock::now();
    Heapsort(Random_c, size);
    auto end2 = chrono::high_resolution_clock::now();
    auto duration2 = chrono::duration_cast<chrono::microseconds>(end2 - start2);
    cout << "Heapsort:" << duration2.count() << " \n";
    cout << "Sorting Correctness: " << (check(Random_c, size) ? "Passed" : "Failed") << "\n";


    vector<int> orderedVec(size);
    for (int i = 0; i < size; ++i) {
        orderedVec[i] = i;
    }
    vector<int> orderedVec_c = orderedVec;
    cout << "testing the order\n";

    auto start3 = chrono::high_resolution_clock::now();
    make_heap(orderedVec.begin(), orderedVec.end());
    sort_heap(orderedVec.begin(), orderedVec.end());
    auto end3 = chrono::high_resolution_clock::now();
    auto duration3 = chrono::duration_cast<chrono::microseconds>(end3 - start3);
    cout << "sort_heap:" << duration3.count() << " \n";
    cout << "Sorting Correctness: " << (check(orderedVec, size) ? "Passed" : "Failed") << "\n";

    auto start4 = chrono::high_resolution_clock::now();
    Heapsort(orderedVec_c, size);
    auto end4 = chrono::high_resolution_clock::now();
    auto duration4 = chrono::duration_cast<chrono::microseconds>(end4 - start4);
    cout << "Heapsort:" << duration4.count() << " \n";
    cout << "Sorting Correctness: " << (check(orderedVec_c, size) ? "Passed" : "Failed") << "\n";
```

图 4: main

```
    vector<int> reversedVec(size);
    for (int i = 0; i < size; ++i) {
        reversedVec[i] = size - i;
    }
    vector<int> reversedVec_c = reversedVec;
    cout << "testing the reversed\n";

    auto start5 = chrono::high_resolution_clock::now();
    make_heap(reversedVec.begin(), reversedVec.end());
    sort_heap(reversedVec.begin(), reversedVec.end());
    auto end5 = chrono::high_resolution_clock::now();
    auto duration5 = chrono::duration_cast<chrono::microseconds>(end5 - start5);
    cout << "sort_heap:" << duration5.count() << " \n";
    cout << "Sorting Correctness: " << (check(reversedVec, size) ? "Passed" : "Failed") << "\n";

    auto start6 = chrono::high_resolution_clock::now();
    Heapsort(reversedVec_c, size);
    auto end6 = chrono::high_resolution_clock::now();
    auto duration6 = chrono::duration_cast<chrono::microseconds>(end6 - start6);
    cout << "Heapsort:" << duration6.count() << " \n";
    cout << "Sorting Correctness: " << (check(reversedVec_c, size) ? "Passed" : "Failed") << "\n";

    vector<int> Random1 = RandomQ(size, numsize1);
    cout << "testing the random\n";
    vector<int> Random1_c = Random1;

    auto start7 = chrono::high_resolution_clock::now();
    make_heap(Random1.begin(), Random1.end());
    sort_heap(Random1.begin(), Random1.end());
    auto end7 = chrono::high_resolution_clock::now();
    auto duration7 = chrono::duration_cast<chrono::microseconds>(end7 - start7);
    cout << "sort_heap:" << duration7.count() << " \n";
    cout << "Sorting Correctness: " << (check(Random1, size) ? "Passed" : "Failed") << "\n";

    auto start8 = chrono::high_resolution_clock::now();
    Heapsort(Random1_c, size);
    auto end8 = chrono::high_resolution_clock::now();
    auto duration8 = chrono::duration_cast<chrono::microseconds>(end8 - start8);
    cout << "Heapsort:" << duration8.count() << " \n";
    cout << "Sorting Correctness: " << (check(Random1_c, size) ? "Passed" : "Failed") << "\n";

    return 0;
```

图 5: main

2 测试的结果

输出为:

```
testing the random
sort_heap:104954
Sorting Correctness: Passed
Heapsort:116553
Sorting Correctness: Passed
testing the order
sort_heap:45427
Sorting Correctness: Passed
Heapsort:62238
Sorting Correctness: Passed
testing the reversed
sort_heap:48152
Sorting Correctness: Passed
Heapsort:55279
Sorting Correctness: Passed
testing the random
sort_heap:81226
Sorting Correctness: Passed
Heapsort:89960
Sorting Correctness: Passed
```

图 6: output

表格为:

	my heapsort time	std::sort_heap time
random sequence	116553	104954
ordered sequence	62238	45427
reverse sequence	55279	48152
repetitive sequence	89960	81226

表 1: the result

3 bug 报告

一切正常。