課題：I111 6th Report　　言語：C++(Console Application)

氏名：GAO, Yuwei

学生番号：s1910092　　提出日：2019/05/29

① ランダムな長さ 100,10000 の数列を作り，P9 バブルソート，P17 シェルソート，P23 ヒープソート，P31 クイックソートを行う．このとき，各コードの“←※”で書かれた部分が実行された回数は何回か，カウントせよ．

長さ ＼ メソッド	Bubble Sort	Shell Sort	Heap Sort	Quick Sort
100	2338 回 0 秒	414 回 0 秒	529 回 0 秒	175 回 0 秒
10000	24052632 回 0.218 秒	151566 回 0.002 秒	118809 回 0.002 秒	33784 回 0.001 秒
100000	1345027527 回 18.871 秒	2618761 回 0.021 秒	1467964 回 0.019 秒	397592 回 0.012 秒

② ランダムな数列の作り方は web や他人を参考にしてよい
186~200 行。Vector、rand()などで実装した。(C++)

③ ソースコードは流用してよい．ソースコードの変更した部分をレポートせよ．配列そのものをレポートしなくてよい．合計Ａ４の２枚程度までにまとめること
すべてのコードを C++にした。追加・変更した部分を次のページに載せた。

④ 長さ 100 から長さ 10000 に増えたときの，回数の増え方が，計算量の観点で予想通りであるか考察せよ．
100、10000、100000 の時の計算量を比較したら、
Bubble Sort：n^2　（ただし長さが増える一方で、計算量増える速度が徐々に減っている。）
Shell Sort：$n^{1.25}$
Heap Sort：$n \log n$
Quick Sort：n
ビッグ O の記法を満たす。

⑤ 余裕があれば，それぞれについて，実行時間を計測して，その増え方が予想通りであるか考察せよ（＋１点）．
計算量のカウントより予想通りである。

```cpp
#include <iostream>
#include <cstring>
#include <stdio.h>
#include <time.h>
#include <vector>
#include <algorithm>
#include <time.h>
template
<
    typename TYPE,
    std::size_t SIZE
>
std::size_t array_length(const TYPE(&)[SIZE])
{
    return SIZE;
}
template
<
    typename TYPE,
    std::size_t SIZE
>
void print(const TYPE(&array)[SIZE])
{
    for (auto i : array)
    {
        std::cout << i << "_";
    }
    std::cout << std::endl;
}
template
<
    typename TYPE,
    typename SIZE
>
void print(const TYPE array, SIZE size)
{
    for (SIZE i = 0; i < size; i++)
    {
        std::cout << array[i] << "_";
    }
    std::cout << std::endl;
}
template
<
    typename TYPE,
    std::size_t SIZE
>
TYPE* bubbleSort(const TYPE(&array)[SIZE])//バブルソート
{
    TYPE* array_ = new TYPE[SIZE];
    memcpy(array_, array, sizeof(array));
    int n = SIZE;long long count = 0;
    for (int k = 1; k < n; k++) {
        for (int i = 0; i < n - k; i++) {
            if (array_[i] > array_[i + 1]) {
                int t = array_[i]; array_[i] = array_[i + 1]; array_[i + 1] = t;
                  count++;
            }
        }
    }
    std::cout << "BubbleSort   " << count << std::endl;
    return array_;
}
template
```

```
64  <
65      typename TYPE,
66      std::size_t SIZE
67  >
68  TYPE* shellSort(const TYPE(&array)[SIZE])//シェルソート
69  {
70      TYPE* array_ = new TYPE[SIZE];
71      memcpy(array_, array, sizeof(array));
72      int n = SIZE, count = 0;
73      for (int gap = n / 2; gap > 0; gap = gap / 2) {
74          for (int i = gap; i < n; i++) {
75              for (int j = i - gap; j >= 0 && array_[j] > array_[j + gap]; j = j - gap)
                    {
76                  auto t = array_[j]; array_[j] = array_[j + gap]; array_[j + gap] = t;
                        count++;
77              }
78          }
79      }
80      std::cout << "ShellSort   " << count << std::endl;
81      return array_;
82  }
83  class HeapArray {
84  private:
85      int* heap;
86      int n;
87  public:
88      int count = 0;
89      HeapArray(int size) {
90          heap = new int[size];
91          n = 0;
92      }
93      void push(int x) {
94          heap[n] = x;
95          int child = n;
96          int parent = (n - 1) / 2;
97          while (child != 0 && x < heap[parent]) {
98              heap[child] = heap[parent]; count++;
99              child = parent;
100             parent = (child - 1) / 2;
101         }
102         heap[child] = x;
103         n++;
104     }
105     int pop() {
106         int minValue = heap[0];
107         heap[0] = heap[n - 1];
108         n--;
109         int parent = 0;
110         int child = parent * 2 + 1;
111         while (child < n) {
112             if (child + 1 < n && heap[child] > heap[child + 1]) child++;
113             if (heap[parent] <= heap[child]) break;
114             int t = heap[child]; heap[child] = heap[parent]; heap[parent] = t; count+
                    +;
115             parent = child;
116             child = parent * 2 + 1;
117         }
118         return minValue;
119     }
120 };
121 template
122 <
123     typename TYPE,
124     std::size_t SIZE
```

```
125 >
126 TYPE* heapSort(const TYPE(&array)[SIZE])//ヒープソート
127 {
128     TYPE* array_ = new TYPE[SIZE];
129     memcpy(array_, array, sizeof(array));
130     int n = SIZE;
131     //std::vector<TYPE> heap;
132     //for (int i = 0; i < SIZE; i++)
133     //{
134     //  heap.push_back(array_[i]);
135     //}
136     //std::make_heap(heap.begin(), heap.end());
137     //for (int i = SIZE - 1; i >= 0; i--)
138     //{
139     //  array_[i] = heap.front();
140     //  std::pop_heap(heap.begin(), heap.end());
141     //  heap.pop_back();
142     //}
143     HeapArray* heap = new HeapArray(n);
144     for (int i = 0; i < n; i++) heap->push(array_[i]);
145     for (int i = 0; i < n; i++) array_[i] = heap->pop();
146     std::cout << "HeapSort   " << heap->count << std::endl;
147     return array_;
148 }
149 int qcount = 0;
150 template
151 <
152     typename TYPE
153 >
154 void qsort(TYPE * array, int left, int right) {
155     if (right <= left) return;
156     int i = left; int j = right; int x = array[(i + j) / 2];
157     while (i <= j) {
158         while (array[i] < x) i = i + 1;
159         while (array[j] > x) j = j - 1;
160         if (i <= j) {
161             TYPE t = array[i]; array[i] = array[j]; array[j] = t; qcount++;
162             i++; j--;
163         }
164     }
165     qsort(array, left, j); qsort(array, i, right);
166 }
167 template
168 <
169     typename TYPE,
170     std::size_t SIZE
171 >
172 TYPE* quickSort(const TYPE(&array)[SIZE])//クイックソート
173 {
174     TYPE* array_ = new TYPE[SIZE];
175     memcpy(array_, array, sizeof(array));
176     int n = SIZE;
177     qsort(array_, 0, n - 1);
178     std::cout << "QuickSort   " << qcount << std::endl;
179     return array_;
180 }
181 template
182 <
183     typename TYPE,
184     std::size_t SIZE
185 >
186 void pushRandomNumbers(TYPE(&array)[SIZE])//ランダムな数列の作り方
187 {
188     std::vector<int> randomNumbers;
```

```
189     for (int i = 0; i < SIZE; i++)
190     {
191         randomNumbers.push_back(i + 1);
192     }
193     for (auto& i : array)
194     {
195         int n = randomNumbers.size();
196         n = rand() % n;
197         i = randomNumbers[n];
198         randomNumbers.erase(randomNumbers.begin() + n);
199     }
200 }
201 int main()
202 {
203     srand((unsigned int)time(NULL));
204     clock_t start[8], finish[8];//実行時間を計測
205     double totaltime[8];
206     int randomArray100[100];
207     int randomArray10000[100000];
208     pushRandomNumbers(randomArray100);
209     pushRandomNumbers(randomArray10000);
210     start[0] = clock();
211     int* bubbleSort100 = bubbleSort(randomArray100);
212     finish[0] = clock();
213     start[1] = clock();
214     int* shellSort100 = shellSort(randomArray100);
215     finish[1] = clock();
216     start[2] = clock();
217     int* heapSort100 = heapSort(randomArray100);
218     finish[2] = clock();
219     start[3] = clock();
220     int* quickSort100 = quickSort(randomArray100);
221     finish[3] = clock();
222     start[4] = clock();
223     int* bubbleSort10000 = bubbleSort(randomArray10000);
224     finish[4] = clock();
225     start[5] = clock();
226     int* shellSort10000 = shellSort(randomArray10000);
227     finish[5] = clock();
228     start[6] = clock();
229     int* heapSort10000 = heapSort(randomArray10000);
230     finish[6] = clock();
231     start[7] = clock();
232     int* quickSort10000 = quickSort(randomArray10000);
233     finish[7] = clock();
234     //print(randomArray10000);
235     //print(bubbleSort10000, array_length(randomArray10000));
236     for (int i = 0; i < 8; i++)
237     {
238         totaltime[i] = ((double)(finish[i] - start[i])) / CLOCKS_PER_SEC;
239         std::cout << totaltime[i] << " Seconds" << std::endl;
240     }
241     return 1;
242 }
243 /*/////////出力結果//////////
244 BubbleSort   2338
245 ShellSort   414
246 HeapSort   529
247 QuickSort   175
248 BubbleSort   24052632
249 ShellSort   151566
250 HeapSort   118809
251 QuickSort   33784
252 0 Seconds
```

```
253  0.001 Seconds
254  0 Seconds
255  0 Seconds
256  0.218 Seconds
257  0.002 Seconds
258  0.002 Seconds
259  0.001 Seconds
260  *//////////////////////////
```