分割實作

壹、 分割

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

template <class Type> inline void exchange(Type* a,Type* b)
  { Type c = *a; *a = *b, *b = c; }
/**
 * @brief array of pointer to use
 * @tparam Type array contain type
 * @tparam Size array length type
 */
template <class Type, class Size> struct Array
  {
    Type* start; Size length;
    Array(Size _length)
      {
        this->start = new Type[_length];
        this->length = _length;
      }
    Array(Type* _start, Size _length)
      {
        this->start = _start;
        this->length = _length;
      }
  };
/**
 * @brief Set random element to a array (Does not include setting random seed) (`std::srand(std::time(nullptr));`)
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array modified target
 */
template <class Type, class Size> inline void setRandomElement_Array(Array<Type, Size>& array)
  { for (size_t i=0; i<array.length; i++) *(array.start+i) = (std::rand()^std::rand())%100; }
template <class Type, class Size> std::ostream& operator<< (std::ostream& outStream, Array<Type, Size>& array)
  {
    outStream<<"[";
    for (size_t i=0; i<array.length; i++)
      {
        if (i) outStream<<", ";
        outStream<<*(array.start+i);
      }
    outStream<<"]";
    return outStream;
  }
```

一、 分割

```cpp
/**
 * @brief Cut Partition. Use the specified index value as the split pivot,
 * move the position of the element so that the position greater than the pivot value is behind the pivot value
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array modified target
 * @param pivot initial pivot position
 * @return end pivot position
 */
template <class Type, class Size> Size singlePivotSplit_Array(Array<Type, Size>& array, Size pivot=0)
  { // std::cout<<"singlePivotSplit_Array: "<<array<<std::endl;
    // Normalized pivot position
    for (; pivot!=0; pivot--) exchange((array.start+pivot), (array.start+pivot-1));
    /**
     * @brief main action: Continue to move all values greater than the pivot value to the back,
     * and finally move the pivot to the front of the first value greater than the pivot value
     * @param i The position retainer of pivot,
     * and elements greater than this position are greater than pivot value
     * @param j Forward explorer,
     * will exchange value with The position retainer of pivot when explored element is greater than pivot value
     */
    for (size_t i, j=i=array.length-1; j<=i && i*j!=0; )
      {
        // The explorer moves The position retainer of pivot forward
        for (; *(array.start+j)>*(array.start+pivot) && i==j && j>0; i=--j) ;
        // The explorer moves to a position greater than the pivot value
        for (; *(array.start+j)<=*(array.start+pivot) && j>0; j--) ;
        // Exchange the value of the explored position to The position retainer of pivot
        if (*(array.start+j)>*(array.start+pivot) && j>0)
          exchange((array.start+j), (array.start+i)), i--, j--;
        // Explore to the end
        if (j==0)
          exchange((array.start+pivot), (array.start+i)), pivot = i;
      }
    return pivot;
  }
```

```cpp
/**
 * @brief Cut Partition. Divide all adjacent 'groupMumber' elements into one group,
 * after each group obtains the value of the specified order, take out all the values again,
 * take out the specified order as the pivot value, and finally use this value for single pivot division.
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array modified target
 * @param groupMumber divide all adjacent 'groupMumber' elements into one group. Automatically initialized to 5.
 * @param groupOrder order of pivot values in each group, Automatically initialized to "SIZE_MAX",
 * If it is greater than groupMumber during execution, it will be automatically regarded as groupMumber/2.
 * @param order initial pivot position, Automatically initialized to "SIZE_MAX",
 * If it is greater than groupMumber during execution, it will be automatically regarded as groupMumber/2.
 * @return end pivot position
 */
template <class Type, class Size> Size multiPivotSplit_Array(Array<Type, Size>& array, Size groupMumber=5, Size groupOrder=SIZE_MAX, Size order=SIZE_MAX)
  { // std::cout<<"multiPivotSplit_Array: "<<array<<std::endl;
    // parameter correction
    if (groupMumber==0 || groupMumber>=array.length) groupMumber = 5;
    if (groupOrder>groupMumber) groupOrder = groupMumber>>1;
    Size g_full=array.length/groupMumber, g_remains=array.length%groupMumber;
    if (order>groupMumber) order = g_full>>1;
    // group infomation
    Array<Type, Size> pivotArray(g_full+(g_remains?1:0));
    // qsort each group of data to get pivot of group
    for (Size g=0; g<pivotArray.length; g++)
      {
        Size g_length = ((g!=g_full)?groupMumber:g_remains);
        std::qsort((array.start+(groupMumber*g)), g_length, sizeof(Type), [](const void* a, const void* b)->
          int{ if ( *(Type*)a <  *(Type*)b ) return -1; if ( *(Type*)a >  *(Type*)b ) return 1; return 0; });
        *(pivotArray.start+g) = *(array.start+(groupMumber*g)+(groupOrder<g_length?groupOrder:(g_length-1)));
      }
    // qsort pivot of group to get pivot of all element
    std::qsort(pivotArray.start, pivotArray.length, sizeof(Type), [](const void* a, const void* b)->
      int{ if ( *(Type*)a <  *(Type*)b ) return -1; if ( *(Type*)a >  *(Type*)b ) return 1; return 0; });
    Type pivot = *(pivotArray.start+order);
    // search pivot index
    Size pivotIndex = groupOrder;
    for (; pivotIndex<array.length; pivotIndex+=groupMumber) if (*(array.start+pivotIndex) == pivot) break;
    if (pivotIndex>=array.length) pivotIndex=pivotIndex-groupOrder+g_remains-1;
    // Use the found pivot index value to perform a single pivot split, and return the split pivot index value
    return singlePivotSplit_Array(array, pivotIndex);
  }
```

```cpp
int main()
  {
    std::srand(std::time(nullptr));
    size_t length; std::cout<<"data length: "; std::cin>>length;
    std::cout<<std::endl;

    struct Array<size_t, size_t> Data(length);
    size_t pivot;

    setRandomElement_Array(Data);
    std::cout<<Data<<std::endl;
    std::cout<<"pivot index: "; std::cin>>pivot;
    std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;
    std::cout<<std::endl; pivot = singlePivotSplit_Array(Data, pivot);
    std::cout<<Data<<std::endl;
    std::cout<<"pivot index: "<<pivot<<std::endl;
    std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;

    std::cout<<std::endl<<std::endl;

    setRandomElement_Array(Data);
    std::cout<<Data<<std::endl;
    size_t groupMumber; std::cout<<"groupMumber: "; std::cin>>groupMumber;
    std::cout<<std::endl; pivot = multiPivotSplit_Array(Data, groupMumber);
    std::cout<<Data<<std::endl;
    std::cout<<"pivot index: "<<pivot<<std::endl;
    std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;
  }
```

二、 快速排序

```
/**
 * @brief quick sort a Array<Type, Size>
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array modified target
 */
template <class Type, class Size> void quickSort_Array(Array<Type, Size>& array)
  {
    // Get segmentation
    Size pivotIndex = multiPivotSplit_Array(array);
    // If less than or equal to the pivot value the part length is greater than 1
    if (pivotIndex>1)
      {
        Array<Type, Size> front(array.start, pivotIndex);
        quickSort_Array(front);
      }
    // If the length of the part greater than the pivot value is greater than 1
    if (array.length-pivotIndex-1>1)
      {
        Array<Type, Size> rear(array.start+pivotIndex+1, array.length-pivotIndex-1);
        quickSort_Array(rear);
      }
  }
```

```
int main()
  {
    std::srand(std::time(nullptr));
    size_t length; std::cout<<"data length: "; std::cin>>length;
    std::cout<<std::endl;

    struct Array<size_t, size_t> Data(length);
    setRandomElement_Array(Data);

    std::cout<<Data<<std::endl;

    std::cout<<std::endl; quickSort_Array(Data);

    std::cout<<Data<<std::endl;
  }
```

三、 找指定大小順序值(從 0 開始)

```cpp
/**
 * @brief find order from a Array<Type, Size>
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array find target
 * @param order order of array
 * @return element values in the specified order
 */
template <class Type, class Size> Type findOrder_Array(Array<Type, Size>& array, Size order)
  {
    // Create a copy
    Array<Type, Size> _array(array.length);
    for (Size i=0; i<_array.length; i++) *(_array.start+i) = *(array.start+i);
    // Get segmentation
    Size pivotIndex = multiPivotSplit_Array(_array);
    // If the target is before the split point
    if (pivotIndex>order)
      {
        Array<Type, Size> front(_array.start, pivotIndex);
        return findOrder_Array(front, order);
      }
    // If the target is after the split point
    else if (pivotIndex<order)
      {
        Array<Type, Size> rear(_array.start+pivotIndex+1, _array.length-pivotIndex-1);
        return findOrder_Array(rear, order-pivotIndex-1);
      }
    // If the split point is the target
    return *(_array.start+pivotIndex);
  }
```

```cpp
int main()
  {
    std::srand(std::time(nullptr));
    size_t length; std::cout<<"data length: "; std::cin>>length;
    std::cout<<std::endl;

    struct Array<size_t, size_t> Data(length);
    setRandomElement_Array(Data);

    std::cout<<Data<<std::endl;

    size_t order; std::cout<<"order: "; std::cin>>order;
    std::cout<<"element: "<<findOrder_Array(Data, order)<<std::endl;

    std::cout<<Data<<std::endl;
  }
```

四、 找中間值(從 0 開始，若於兩元素之間，回傳最小的元素)

```cpp
/**
 * @brief find middle element from a Array<Type, Size>
 * @tparam Type array contain type
 * @tparam Size array length type
 * @param array find target
 * @return middle element values from array
 */
template <class Type, class Size> Type findMiddle_Array(Array<Type, Size>& array)
  { // return findOrder_Array(array, array.length>>1);
    // Create a copy
    Array<Type, Size> _array(array.length);
    for (Size i=0; i<_array.length; i++) *(_array.start+i) = *(array.start+i);
    Size order = _array.length>>1;
    // Get segmentation
    Size pivotIndex = multiPivotSplit_Array(_array);
    // If the target is before the split point
    if (pivotIndex>order)
      {
        Array<Type, Size> front(_array.start, pivotIndex);
        return findOrder_Array(front, order);
      }
    // If the target is after the split point
    else if (pivotIndex<order)
      {
        Array<Type, Size> rear(_array.start+pivotIndex+1, _array.length-pivotIndex-1);
        return findOrder_Array(rear, order-pivotIndex-1);
      }
    // If the split point is the target
    return *(_array.start+pivotIndex);
  }
```

```cpp
int main()
  {
    std::srand(std::time(nullptr));
    size_t length; std::cout<<"data length: "; std::cin>>length;
    std::cout<<std::endl;

    struct Array<size_t, size_t> Data(length);
    setRandomElement_Array(Data);

    std::cout<<Data<<std::endl;

    std::cout<<"middle element: "<<findMiddle_Array(Data)<<std::endl;

    std::cout<<Data<<std::endl;
  }
```