Zadanie 1: zaimplementuj podane poniżej algorytmy sortowania:

1. Sortowanie przez proste wybieranie:

```
Require: tablica A o rozmiarze n
{A=[0,. . . ,n-1]}
1: for all i = 0 to n − 2 do
2:         min = i;
3:         for all j = i + 1 to n − 1 do
4:         if A[j] < A[min] then
5:         min = j;
6:         end if
7:         j = j + 1;
8:         end for
           {zamiana elementu A[min] z elemente
           w  tablicy A}
9:         zamiana (A, min, i);
10:        i = i + 1;
11: end for
```

2. Sortowanie przez wstawianie:

```
Require: tablica A o rozmiarze n
{A=[0,. . . ,n-1]}
1: for i = 1 to n − 1 do
2:         key = A[i];
3:         j = i − 1;
4:         while j ≥ 0 and A[j] > key d
5:         A[j + 1] = A[j];
6:         j = j − 1;
7:         end while
8:         A[j + 1] = key ;
9: end for
```

3. Sortowanie bąbelkowe:

```
Require: tablica A o rozmiarze
{A=[0,. . . ,n-1]}
1: for i = 0 to n − 1 do
2:  for j = n downto i + 1 do
3:   if A[j − 1] > A[j] then
4:     Zamiana(A, j − 1, j);
5:   end if
6:  end for
7: end for
```

4. Sortowanie szybkie:
Funkcja PARTITION:

```
Require:  tablica A[p..q]
1:  PARTITION(A, p, q)
2:  x = A[p]; {element osiowy = A[p]}
3:  i = p;
4:  for all j = p + 1 to q do
5:     if A[j] ≤ x then
6:        i = i + 1;
7:        zamien(A[i],A[j]);
8:     end if
9:  end for
10: zamien (A[p], A[i]);
11: return  i
```

samo sortowanie szybkie:

QuickSort (A, p, r)

- Jeśli $p = r$, to koniec
- Jeśli $p < r$, to
  - q = PARTITION(A, p, r).
  - QuickSort(A, p, q-1).
  - QuickSort(A, q+1, r).

Wywołanie: QuickSort(A, 1, n)

5. Sortowanie przez scalanie:

MergeSort A[1..n]

- Jeśli n = 1, to koniec.
- Jeśli n ≥2, rekurencyjnie posortuj A[1..n/2] i A[n/2+1..n].
- Scal obie połowy A w jedna posortowana tablicę.

Na następnej stronie znajdują się fragmenty implementacji tych algorytmów w C/C++. Gotowe programy w wybranym języku należy przesłać na moodle. Plik liczby.zip zawiera losowe liczby (rand.txt) oraz te same liczby, tylko posortowane (sort.txt), więc można używać tych liczb do testowania swoich programów.

1. Sortowanie przez proste wybieranie:

```
void swap (int data[], int i, int j)
{
  int temp = data[i];
  data[i] = data[j];
  data[j] = temp;
}

int max(int[] a, int n)
{
  int currentMax = 0;
  for (int i = 1; i <= n; i++)
    if (a[currentMax] < a[i]) currentMax = i;
  return currentMax;
}


void selectionSort(int[] a, int size)
{
  for (int n = size; n > 1; n--)
  {
    int j = max(a, n-1);
    swap(a, j, n - 1);
  }
}
```

2. Sortowanie przez wstawianie:

```
void insert(int[] a, int n, int x)
{
  // insert x into a[0..i-1]
  int j;
  for (j = i - 1; j >= 0 && x < a[j]; j--)
      a[j + 1] = a[j];
  a[j + 1] = x;
}



void insertSort(int[] a, int n)
{
  for (int i = 1; i < n; i++)
  {
    // insert a[i] into a[0:i-1]
    insert(a, i, a[i]);
  }
}
```

3. Sortowanie bąbelkowe:

```
void bubbleSort(int A[],int n)
{
  for(int i=0; i<n; i++)
    for(int j=n-1; j>i; j--)
      if (A[j] < A[j-1]) {
        //zamiana A[j-1] z A[j
        int temp= A[j-1];
        A[j-1] = A[j];
        A[j]=temp;
      }
}
```

4. Sortowanie szybkie:

```
void quickSort (int a[], int left, int right) {
  if (left < right){
    // podział tablicy
    int m = left;
    for (int k = left + 1; k <= right; ++k) {
        if (a[k] < a[left]) swap(a[++m], a[k]);
    }
    swap(a[left], a[m]);
    //Rekurencja
    quickSort(a, left, m - 1);
    quickSort(a, m + 1, right);
 }
}
void swap(int* x, int* y) {
  int tmp = *x;  *x = *y;  *y = tmp;
}
```

5. Sortowanie przez scalanie:

```
void merge(T* a, int left, int mid, int right)
{
  int *t = calloc(right+1,sizeof(int));
  int n = right - left + 1;
  int i = left, j = mid + 1, k = 0;
  while (i <= mid && j <= right) {
    if (a[i] < a[j] )  t[k++] = a[i++];
    else t[k++] = a[j++];
  }
  while (i <= mid) { // Dolaczanie koncowki pierwszej podtablicy
    t[k++] = a[i++];
  }
  while (j <= right) {// Dolaczanie koncowki drugiej podtablicy
    t[k++] = a[j++];
  }
  // Kopiowanie tablicy pomocniczej
  for (k = 0; k < n; ++k)  a[left + k] = t[k];
  free(t)
}
```

```
MergeSort(int A[], int left, int right)
{
  if (right>left) {
    mid = (left + right) / 2;
    MergeSort(A, left, mid);
    MergeSort(A, mid +1, right);
    merge (A, left, mid, right)
  }
}
```