**Insertie**

```
void insertionSort(int numbers[], int array_size)
{
	int i, j, index;

	for (i = 1; i < array_size; i++)
	{
		index = numbers[i];
		j = i;
		while ((j > 0) && (numbers[j - 1] > index))
		{
			numbers[j] = numbers[j - 1];
			j = j - 1;
		}
		numbers[j] = index;
	}
}
```

```
8 2 10 5 1 3 7 15 4 11

i = 1,index = 2
j = 1 --> j = 0 -->     8 8 10 5 1 3 7 15 4 11
2 8 10 5 1 3 7 15 4 11

i = 2,index = 10
2 8 10 5 1 3 7 15 4 11

i = 3,index = 5
j = 3 --> j = 2 -->     2 8 10 10 1 3 7 15 4 11
j = 2 --> j = 1 -->     2 8 8 10 1 3 7 15 4 11
2 5 8 10 1 3 7 15 4 11

i = 4,index = 1
j = 4 --> j = 3 -->     2 5 8 10 10 3 7 15 4 11
j = 3 --> j = 2 -->     2 5 8 8 10 3 7 15 4 11
j = 2 --> j = 1 -->     2 5 5 8 10 3 7 15 4 11
j = 1 --> j = 0 -->     2 2 5 8 10 3 7 15 4 11
1 2 5 8 10 3 7 15 4 11

i = 5,index = 3
j = 5 --> j = 4 -->     1 2 5 8 10 10 7 15 4 11
j = 4 --> j = 3 -->     1 2 5 8 8 10 7 15 4 11
j = 3 --> j = 2 -->     1 2 5 5 8 10 7 15 4 11
1 2 3 5 8 10 7 15 4 11

i = 6,index = 7
j = 6 --> j = 5 -->     1 2 3 5 8 10 10 15 4 11
j = 5 --> j = 4 -->     1 2 3 5 8 8 10 15 4 11
1 2 3 5 7 8 10 15 4 11

i = 7,index = 15
1 2 3 5 7 8 10 15 4 11

i = 8,index = 4
j = 8 --> j = 7 -->     1 2 3 5 7 8 10 15 15 11
j = 7 --> j = 6 -->     1 2 3 5 7 8 10 10 15 11
```

```
j = 6 --> j = 5 -->     1 2 3 5 7 8 8 10 15 11
j = 5 --> j = 4 -->     1 2 3 5 7 7 8 10 15 11
j = 4 --> j = 3 -->     1 2 3 5 5 7 8 10 15 11
1 2 3 4 5 7 8 10 15 11

i = 9,index = 11
j = 9 --> j = 8 -->     1 2 3 4 5 7 8 10 15 15
1 2 3 4 5 7 8 10 11 15
```

**Bubble**

```
void bubbleSort(int numbers[], int array_size)
{
	int i, j, temp;
	for (i = (array_size - 1); i >= 0; i--)
	{
		for (j = 1; j <= i; j++)
		{
			if (numbers[j - 1] > numbers[j])
			{
				temp = numbers[j - 1];
				numbers[j - 1] = numbers[j];
				numbers[j] = temp;
			}
		}

	}
}
```

**8 2 10 5 1 3 7 15 4 11**

```
i = 9
j = 1 -->    2 8 10 5 1 3 7 15 4 11
j = 2 -->    2 8 10 5 1 3 7 15 4 11
j = 3 -->    2 8 5 10 1 3 7 15 4 11
j = 4 -->    2 8 5 1 10 3 7 15 4 11
j = 5 -->    2 8 5 1 3 10 7 15 4 11
j = 6 -->    2 8 5 1 3 7 10 15 4 11
j = 7 -->    2 8 5 1 3 7 10 15 4 11
j = 8 -->    2 8 5 1 3 7 10 4 15 11
j = 9 -->    2 8 5 1 3 7 10 4 11 15
2 8 5 1 3 7 10 4 11 15
```

```
i = 8
j = 1 -->    2 8 5 1 3 7 10 4 11 15
j = 2 -->    2 5 8 1 3 7 10 4 11 15
j = 3 -->    2 5 1 8 3 7 10 4 11 15
j = 4 -->    2 5 1 3 8 7 10 4 11 15
j = 5 -->    2 5 1 3 7 8 10 4 11 15
j = 6 -->    2 5 1 3 7 8 10 4 11 15
j = 7 -->    2 5 1 3 7 8 4 10 11 15
j = 8 -->    2 5 1 3 7 8 4 10 11 15
2 5 1 3 7 8 4 10 11 15
```

```
i = 7
j = 1 -->    2 5 1 3 7 8 4 10 11 15
j = 2 -->    2 1 5 3 7 8 4 10 11 15
j = 3 -->    2 1 3 5 7 8 4 10 11 15
j = 4 -->    2 1 3 5 7 8 4 10 11 15
j = 5 -->    2 1 3 5 7 8 4 10 11 15
j = 6 -->    2 1 3 5 7 4 8 10 11 15
j = 7 -->    2 1 3 5 7 4 8 10 11 15
2 1 3 5 7 4 8 10 11 15
```

```
i = 6
j = 1 -->   1 2 3 5 7 4 8 10 11 15
j = 2 -->   1 2 3 5 7 4 8 10 11 15
j = 3 -->   1 2 3 5 7 4 8 10 11 15
j = 4 -->   1 2 3 5 7 4 8 10 11 15
j = 5 -->   1 2 3 5 4 7 8 10 11 15
j = 6 -->   1 2 3 5 4 7 8 10 11 15
1 2 3 5 4 7 8 10 11 15

i = 5
j = 1 -->   1 2 3 5 4 7 8 10 11 15
j = 2 -->   1 2 3 5 4 7 8 10 11 15
j = 3 -->   1 2 3 5 4 7 8 10 11 15
j = 4 -->   1 2 3 4 5 7 8 10 11 15
j = 5 -->   1 2 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15

i = 4
j = 1 -->   1 2 3 4 5 7 8 10 11 15
j = 2 -->   1 2 3 4 5 7 8 10 11 15
j = 3 -->   1 2 3 4 5 7 8 10 11 15
j = 4 -->   1 2 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15

i = 3
j = 1 -->   1 2 3 4 5 7 8 10 11 15
j = 2 -->   1 2 3 4 5 7 8 10 11 15
j = 3 -->   1 2 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15

i = 2
j = 1 -->   1 2 3 4 5 7 8 10 11 15
j = 2 -->   1 2 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15

i = 1
j = 1 -->   1 2 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15

i = 0
1 2 3 4 5 7 8 10 11 15
```

**Selectie**

```
void selectionSort(int numbers[], int array_size)
{
	int i, j;
	int min, temp;

	for (i = 0; i < array_size - 1; i++)
	{
		min = i;
		for (j = i + 1; j < array_size; j++)
		{
			if (numbers[j] < numbers[min])
				min = j;
		}

		temp = numbers[i];
		numbers[i] = numbers[min];
		numbers[min] = temp;

	}
}
```

**8 2 10 5 1 3 7 15 4 11**

i = 0 ind min = 4
1 **2 10 5 8 3 7 15 4 11**

i = 1 ind min = 1
1 2 **10 5 8 3 7 15 4 11**

i = 2 ind min = 5
1 2 3 **5 8 10 7 15 4 11**

i = 3 ind min = 8
1 2 3 4 **8 10 7 15 5 11**

i = 4 ind min = 8
1 2 3 4 5 **10 7 15 8 11**

i = 5 ind min = 6
1 2 3 4 5 7 **10 15 8 11**

i = 6 ind min = 8
1 2 3 4 5 7 8 **15 10 11**

i = 7 ind min = 8
1 2 3 4 5 7 8 10 **15 11**

i = 8 ind min = 9
1 2 3 4 5 7 8 10 11 **15**

**Quick**

```
void q_sort(int numbers[], int left, int right)
{
	int pivot, i, j;

	pivot = left;
	i = left;   // primul element
	j = right;  // ultimul element

	if (left < right)
	{
		while (i < j)
		{
			while ((numbers[i] <= numbers[pivot]) && i < right) i++;

			while ((numbers[j] >= numbers[pivot]) && j > left) j--;

			//interschimba numbers[i] cu numbers[j]
			if (i < j)
			{
				int tmp = numbers[i];
				numbers[i] = numbers[j];
				numbers[j] = tmp;
			}

		}

		// i>=j
		// interschimba pivot cu numbers[j]
			int tmp = numbers[j];
			numbers[j] = numbers[pivot];
			numbers[pivot] = tmp;

		q_sort(numbers, left, j - 1);
		q_sort(numbers, j + 1, right);

	}
}
```

**8** 2 10 5 1 3 7 15 4 11

ind_left = 0; ind_right = 9; ind_pivot = 0;
i = 2;  j = 8;  8 2 4 5 1 3 7 15 10 11
i = 7;  j = 6;  8 2 4 5 1 3 7 15 10 11
7 2 4 5 1 3 **8** 15 10 11
Quick

**7** 2 4 5 1 3 8 15 10 11

ind_left = 0; ind_right = 5; ind_pivot = 0;
i = 5;  j = 5;  7 2 4 5 1 3 8 15 10 11
3 2 4 5 1 **7** 8 15 10 11
Quick

**3** 2 4 5 1 7 8 15 10 11

ind_left = 0; ind_right = 4; ind_pivot = 0;
i = 2;  j = 4;  3 2 1 5 4 7 8 15 10 11
i = 3;  j = 2;  3 2 1 5 4 7 8 15 10 11
**1 2 3** 5 4 7 8 15 10 11
Quick

**1** 2 3 5 4 7 8 15 10 11

ind_left = 0; ind_right = 1; ind_pivot = 0;
i = 1;  j = 0;  1 2 3 5 4 7 8 15 10 11
**1** 2 3 5 4 7 8 15 10 11
Quick

1 2 3 5 4 7 8 15 10 11

ind_left = 0; ind_right = -1; ind_pivot = 0;
Quick

1 2 3 5 4 7 8 15 10 11

ind_left = 1; ind_right = 1; ind_pivot = 1;
Quick

1 2 3 5 **4** 7 8 15 10 11

ind_left = 3; ind_right = 4; ind_pivot = 3;
i = 4;  j = 4;  1 2 3 5 4 7 8 15 10 11
1 2 3 4 5 7 8 15 10 11
Quick

1 2 3 **4** 5 7 8 15 10 11

ind_left = 3; ind_right = 3; ind_pivot = 3;
Quick

1 2 3 4 5 7 8 15 10 11

ind_left = 5; ind_right = 4; ind_pivot = 5;
Quick

1 2 3 4 5 7 8 15 10 11

ind_left = 6; ind_right = 5; ind_pivot = 6;
Quick

1 2 3 4 5 7 8 **15** 10 11

ind_left = 7; ind_right = 9; ind_pivot = 7;
i = 9;  j = 9;  1 2 3 4 5 7 8 15 10 11
1 2 3 4 5 7 8 11 10 **15**
Quick

1 2 3 4 5 7 8 **11** 10 15

ind_left = 7; ind_right = 8; ind_pivot = 7;
i = 8;  j = 8;  1 2 3 4 5 7 8 11 10 15

```
1 2 3 4 5 7 8 10 11 15
Quick

1 2 3 4 5 7 8 10 11 15

ind_left = 7; ind_right = 7; ind_pivot = 7;
Quick

1 2 3 4 5 7 8 10 11 15

ind_left = 9; ind_right = 8; ind_pivot = 9;
Quick

1 2 3 4 5 7 8 10 11 15

ind_left = 10; ind_right = 9; ind_pivot = 10;
```

**Heap**

```
void heapSort(int numbers[], int array_size)
{
	int i, temp,n;

	n = array_size; // dimensiunea arborelui heap curent
	while (n > 1)
	{
	// determina arborele heap
	for (i = (n / 2); i >= 1; i--)
		heap_tree(numbers, i, n);

		// extrage radacina
		temp = numbers[0];
		numbers[0] = numbers[n-1];
		numbers[n-1] = temp;
		n--; // dimensiunea arborelui se reduce cu 1

	}
}

void heap_tree(int numbers[], int root, int bottom)
// root - numarul nodului de la 1,2,...N; bottom - dimensiunea sirului = N
{
	int done, maxChild, temp;
	// maxChild = 1, 2,... pozitia nodului de valoare maxima
	done = 0;
	while ((root * 2 <= bottom) && (!done)) // nodul exista
	{
		if (root * 2 == bottom)
		// nodul are doar 1 descendent (doar pe stinga e posibil,
		// altfel are 2 descendenti)
			maxChild = root * 2 ;
		else if (numbers[root * 2 - 1] > numbers[root * 2 + 1 - 1])
			maxChild = root * 2;
		else
			maxChild = root * 2 + 1;

		if (numbers[root-1] < numbers[maxChild-1])
		{
			temp = numbers[root-1];
			numbers[root-1] = numbers[maxChild-1];
			numbers[maxChild-1] = temp;
			root = maxChild;
		}
		else
			done = 1;
	}
}
```

```
8 2 10 5 1 3 7 15 4 11

Heap curent --> 15 11 10 5 8 3 7 2 4 1
1 11 10 5 8 3 7 2 4 15
Heap curent --> 11 8 10 5 1 3 7 2 4 15
```

**4 8 10 5 1 3 7 2** 11 15
Heap curent --> **10 8 7 5 1 3 4 2** 11 15
**2 8 7 5 1 3 4** 10 11 15
Heap curent --> **8 5 7 2 1 3 4** 10 11 15
**4 5 7 2 1 3** 8 10 11 15
Heap curent --> **7 5 4 2 1 3** 8 10 11 15
**3 5 4 2 1** 7 8 10 11 15
Heap curent --> **5 3 4 2 1** 7 8 10 11 15
**1 3 4 2** 5 7 8 10 11 15
Heap curent --> **4 3 1 2** 5 7 8 10 11 15
**2 3 1** 4 5 7 8 10 11 15
Heap curent --> **3 2 1** 4 5 7 8 10 11 15
**1 2** 3 4 5 7 8 10 11 15
Heap curent --> **2 1** 3 4 5 7 8 10 11 15
1 2 3 4 5 7 8 10 11 15