**Задание 1**

Выполнить сравнительный анализ сортировки слиянием и  
сортировки Хоара.

**Сортировка Хоара**

**Cортировка слиянием**

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

**Анализ сложности не рекурсивной функции merge**

void merge(int arr[], int l, int m, int r)

{

int i, j, k; |

int n1 = m - l + 1; |

int n2 = r - m; | c1 1 раз

|

int L[n1], R[n2]; |

for (i = 0; i < n1; i++) | c2 n/2 раз

L[i] = arr[l + i]; |

for (j = 0; j < n2; j++) | c3 n/2 раз

R[j] = arr[m + 1+ j]; |

i = 0; |

j = 0; | c4 1 раз

k = l; |

while (i < n1 && j < n2) | c5 t раз

{

if (L[i] <= R[j]) |

{ | c6 m раз

arr[k] = L[i]; |

i++; |

}

else |

{ | c7 t-m раз

arr[k] = R[j]; |

j++; |

}

k++; |

}

while (i < n1) |

{ |

arr[k] = L[i]; | c8 n/2-m раз

i++; |

k++; |

} |

while (j < n2) |

{ |

arr[k] = R[j]; | c9 n/2-(t-m) раз

j++; |

k++; |

} |

}

при *t* = *n* (и, следовательно, m = *n / 2*) =>

при *t* = *n / 2*, в этом случае *m* может быть *0* или *n / 2*.

1) При *m* = 0

2) При *m* = *n / 2*

Исходя из лучшего и из худшего случаев, можно сделать вывод, что

**Анализ сложности рекурсивной функции mergeSort**

При *n* = 1, условие if (l < r) сразу же не выполнится и алгоритм завершится. Тогда .

В остальных случаях выполнится тело условия

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2; | Вход Θ(1)

mergeSort(arr, l, m); | Рекурсивные вызовы 2 \* T(n/2)

mergeSort(arr, m+1, r); |

merge(arr, l, m, r); | Выход Θ(n)

}

}

Метод итераций

|  |  |  |
| --- | --- | --- |
|  | Быстрая сортировка | Сортировка слиянием |
| Худший случай |  |  |
| Лучший случай |  |  |
| Средний случай |  |  |

**Задание 2**

Выполнить анализ временной сложности рекурсивной реализации  
бинарного поиска.

int binarySearch(int\* arr, int left, int right, int key) {

if (right == left) { |

return arr[right] == key ? right : -1; | Вход Θ(1)

} |

int middle = (left + right) / 2; |

if (key < arr[middle]) { |

return binarySearch(arr, left, middle - 1); |

} else if (key > arr[middle]) { | Рекурсивный вызов

return binarySearch(arr, middle + 1, right); | T(n/2), либо Θ(1)

} else { | если элемент

return middle; | найден

} |

}

**Задание 3**

Оценить временную сложность рекурсивного алгоритма двумя  
способами.

*Procedure Soch (i : Integer);  
Var k : Integer;  
Begin  
 If i>n Then Print(a)  
 Else For k:=1 To n Do  
 Begin  
 a[i]:=k;  
 Soch(i+1);  
 End;  
End;*

В худшем случае: при начальном *i* = 1

В лучшем случае: при начальном *i > n*