МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ

Інститут комп’ютерних систем

Кафедра інформаційних систем

Звіт

Лабораторна робота №2

З дисципліни **«**Теорія алгоритмів**»**

**Тема:** **«Логарифмічні алгоритми сортування»­**

Виконав:

Студент групи AI-191У

Ігнатюк А.Г

Перевірили:

Смик С. Ю

Одеса ­­­­– 2020

**Тема:**  «Логарифмічні алгоритми сортування»

**Мета роботи:** Розглянути різновиди логарифмічних алгоритмів сортування: злиття та швидка. Порівняти використані алгоритми.

**План:**

1. Сортування алгоритмом злиття
2. Сортування алгоритмом швидкого сортування
3. Порівняння алгоритмів
4. Висновок
5. **Сортировка слиянием**

Ниже представлен псевдокод данного алгоритма:

**array = [28, 76, 27, 10, 5, 35, 95, 16, 33]** # defining an array

**left = 0**

**right = len(array)**

**def mergeSort(array, left, right) :** # main function

**if (****(left+1) >= right) : return** # if array can’t be divided anymore just skip

**mid = (left+right)//2** # calculating the index of middle element

**mergeSort(array, left, mid)** # recursive call to sort the left side

**mergeSort(array, mid, right)** # recursive call to sort the right side

**merge(array, left, mid, right)** # combining two sorted subarrays in one solid array

**def merge(a, left, mid, right) :** # function to combine and sort arrays

**it1 = 0**

**it2 = 0**

**result = []** # addition memory to keep results of sorting

# main cycle to compare elements of two subarrays

**while (left+it1 < mid and mid+it2 < right) :**

# if element of left side is smaller than element of right side insert the left one to resulting array

**if (a[left+it1] < a[mid+it2]) :**

**result.insert(it1+it2, a[left+it1])**

**it1 += 1**

**else :** # else insert the right side element first

**result.insert(it1+it2, a[mid+it2])**

**it2 += 1**

# additional cycles to insert values

**while (left+it1 < mid) :**

**result.insert(it1+it2, a[left+it1])**

**it1 += 1**

**while (mid+it2 < right) :**

**result.insert(it1+it2, a[mid+it2])**

**it2 += 1**

# copying resulting array to the source

**for i in range(0, (it1+it2)) :**

**a[left + i] = result[i]**

**mergeSort(array, left, right)**

**for i in array :**

**print(i, end=" ")**

Пример численного моделирования приведенного псевдокода рекурсивного алгоритма:

Исходные данные array = [28, 76, 27, 10, 5, 35, 95, 16, 33] n=9

left = 0, right = n

1. mergeSort(array, left, right) [left=0, right=9]

if ((left+1) >= right) [1>=9] – FALSE

mid = (left+right)//2 [mid=4]

2. mergeSort(array, left, mid) [left=0,mid=4]

if ((left+1) >= right) [1>=5] – FALSE

mid = (left+right)//2 [mid=2]

3. mergeSort(array, left, mid) [left=0,mid=2]

if ((left+1) >= right) [1>=2] – FALSE

mid = (left+right)//2 [mid=1]

4. mergeSort(array, left, mid) [left=0,mid=1]

if ((left+1) >= right) [1>=1] – TRUE

return

5. mergeSort(array, mid, right) [mid=1, right=2]

if ((left+1) >= right) [2>=2] – TRUE

return

6. merge(array, left, mid, right) [left=0, mid=1, right=2]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [0<1 && 1<2] – TRUE

if (a[left+it1] < a[mid+it2]) [28<76] – TRUE

result.insert(it1+it2, a[left+it1]) [result[0] = 28]]

it1 += 1 [it1=1]

while (left+it1 < mid and mid+it2 < right) [1<1 && 1<2] – FALSE

while (left+it1 < mid) [1<1] – FALSE

while (mid+it2 < right) [0 < 2] – TRUE

result.insert(it1+it2, a[mid+it2]) [result[1] = 76]

it2 += 1 [it2=1]

for i in range(0, (it1+it2)) [from 0 to 2-1]

a[left + i] = result[i] [a[0] = 28]

a[left + i] = result[i] [a[1] = 76]

[28, 76, 27, 10, 5, 35, 95, 16, 33]

7. mergeSort(array, mid, right) [mid=2, right=4]

if ((left+1) >= right) [3>=4] – FALSE

mid = (left+right)//2 [mid=3]

8. mergeSort(array, left, mid) [left=2, mid=3]

if ((left+1) >= right) [3>=3] – TRUE

return

9. mergeSort(array, mid, right) [mid=3, right=4]

if ((left+1) >= right) [4>=4] – TRUE

return

10. merge(array, left, mid, right) [left=2, mid=3, right=4]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [2<3 && 3<4] – TRUE

if (a[left+it1] < a[mid+it2]) [27<10] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[0] = 10]

it2 += 1 [it2=1]

while (left+it1 < mid and mid+it2 < right) [2<3 && 4<4] – FALSE

while (left+it1 < mid) [2<3] – TRUE

result.insert(it1+it2, a[left+it1]) [result[1] = 27]

it1 += 1 [it1=1]

while (left+it1 < mid) [3<3] – FALSE

while (mid+it2 < right) [4<4] – FALSE

for i in range(0, (it1+it2)) [from 0 to 2-1]

a[left + i] = result[i] [a[2] = 10]

a[left + i] = result[i] [a[3] = 27]

[28, 76, 10, 27, 5, 35, 95, 16, 33]

11. merge(array, left, mid, right) [left=0, mid=2, right=4]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [0<2 && 2<4] – TRUE

if (a[left+it1] < a[mid+it2]) [28<10] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[0] = 10]

it2 += 1 [it2=1]

while (left+it1 < mid and mid+it2 < right) [0<2 && 3<4] – TRUE

if (a[left+it1] < a[mid+it2]) [28<27] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[1] = 27]

it2 += 1 [it2=2]

while (left+it1 < mid and mid+it2 < right) [0<2 && 4<4] – FALSE

while (left+it1 < mid) [0 < 2] – TRUE

result.insert(it1+it2, a[left+it1]) [result[2] = 28]

it1 += 1 [it1=1]

while (left+it1 < mid) [0 < 2] – TRUE

result.insert(it1+it2, a[left+it1]) [result[3] = 76]

it1 += 1 [it1=2]

while (mid+it2 < right) [4<4] – FALSE

for i in range(0, (it1+it2)) [from 0 to 4-1]

a[left + i] = result[i] [a[0] = 10]

a[left + i] = result[i] [a[1] = 27]

a[left + i] = result[i] [a[2] = 28]

a[left + i] = result[i] [a[3] = 76]

[10, 27, 28, 76, 5, 35, 95, 16, 33]

12. mergeSort(array, mid, right) [mid=4, right=9]

if ((left+1) >= right) [5>=9] – FALSE

mid = (left+right)//2 [mid=6]

13. mergeSort(array, left, mid) [left=4, mid=6]

if ((left+1) >= right) [5>=6] – FALSE

mid = (left+right)//2 [mid=5]

14. mergeSort(array, left, mid) [left=4, mid=5]

if ((left+1) >= right) [5>=5] – TRUE

return

15. mergeSort(array, mid, right) [mid=5, right=6]

if ((left+1) >= right) [6>=6] – TRUE

return

16. merge(array, left, mid, right) [left=4, mid=5, right=6]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [4<5 && 5<6] – TRUE

if (a[left+it1] < a[mid+it2]) [5<35] – TRUE

result.insert(it1+it2, a[left+it1]) [result[0]=5]

it1 += 1 [it1=1]

while (left+it1 < mid and mid+it2 < right) [5<5 && 5<6] – FALSE

while (left+it1 < mid) [5<5] – FALSE

while (mid+it2 < right) [5<6] – TRUE

result.insert(it1+it2, a[mid+it2]) [result[1]=35]

it2 += 1 [it2=1]

while (mid+it2 < right) [5<6] – FALSE

for i in range(0, (it1+it2)) [from 0 to 2-1]

a[left + i] = result[i] [a[4] = 5]

a[left + i] = result[i] [a[5] = 35]

[10, 27, 28, 76, 5, 35, 95, 16, 33]

17. mergeSort(array, mid, right) [mid=6, right=9]

if ((left+1) >= right) [7>=9] – FALSE

mid = (left+right)//2 [mid=7]

18. mergeSort(array, left, mid) [left=6, mid=7]

if ((left+1) >= right) [7>=7] – TRUE

return

19. mergeSort(array, mid, right) [mid=7, right=9]

if ((left+1) >= right) [8>=9] – FALSE

mid = (left+right)//2 [mid=8]

20. mergeSort(array, left, mid) [left=7, mid=8]

if ((left+1) >= right) [8>=8] – TRUE

return

21. mergeSort(array, mid, right) [mid=8, right=9]

if ((left+1) >= right) [9>=9] – TRUE

return

22. merge(array, left, mid, right) [left=7, mid=8, right=9]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [7<8 && 8<9] – TRUE

if (a[left+it1] < a[mid+it2]) [16<33] – TRUE

result.insert(it1+it2, a[left+it1]) [result[0]=16]

it1 += 1 [it1=1]

while (left+it1 < mid and mid+it2 < right) [8<8 && 8<9] – FALSE

while (left+it1 < mid) [8<8] – FALSE

while (mid+it2 < right) [8<9] – TRUE

result.insert(it1+it2, a[mid+it2]) [result[1]=33]

it2 += 1 [it2=1]

while (mid+it2 < right) [9<9] – FALSE

for i in range(0, (it1+it2)) [from 0 to 2-1]

a[left + i] = result[i] [a[7] = 16]

a[left + i] = result[i] [a[8] = 33]

[10, 27, 28, 76, 5, 35, 95, 16, 33]

23. merge(array, left, mid, right) [left=6, mid=7, right=9]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [6<7 && 7<9] – TRUE

if (a[left+it1] < a[mid+it2]) [95<16] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[0]=16]

it2 += 1 [it2=1]

while (left+it1 < mid and mid+it2 < right) [6<7 && 8<9] – TRUE

if (a[left+it1] < a[mid+it2]) [95<33] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[1]=33]

it2 += 1 [it2=2]

while (left+it1 < mid and mid+it2 < right) [6<7 && 9<9] – FALSE

while (left+it1 < mid) [6<7] – TRUE

result.insert(it1+it2, a[left+it1]) [result[2] = 95]

it1 += 1 [it1=1]

while (left+it1 < mid) [7<7] – FALSE

while (mid+it2 < right) [9<9] – FALSE

for i in range(0, (it1+it2)) [from 0 to 3-1]

a[left + i] = result[i] [a[6] = 16]

a[left + i] = result[i] [a[7] = 33]

a[left + i] = result[i] [a[8] = 95]

[10, 27, 28, 76, 5, 35, 16, 33, 95]

24. merge(array, left, mid, right) [left=4, mid=6, right=9]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [4<6 && 6<9] – TRUE

if (a[left+it1] < a[mid+it2]) [5<16] – TRUE

result.insert(it1+it2, a[left+it1]) [result[0]=5]

it1 += 1 [it1=1]

while (left+it1 < mid and mid+it2 < right) [5<6 && 6<9] – TRUE

if (a[left+it1] < a[mid+it2]) [35<16] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[1]=16]

it2 += 1 [it2=1]

while (left+it1 < mid and mid+it2 < right) [5<6 && 7<9] – TRUE

if (a[left+it1] < a[mid+it2]) [35<33] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[2]=33]

it2 += 1 [it2=2]

while (left+it1 < mid and mid+it2 < right) [5<6 && 8<9] – TRUE

if (a[left+it1] < a[mid+it2]) [35<95] – TRUE

result.insert(it1+it2, a[left+it1]) [result[3]=35]

it1 += 1 [it1=2]

while (left+it1 < mid and mid+it2 < right) [6<6 && 8<9] – FALSE

while (left+it1 < mid) [6<6] – FALSE

while (mid+it2 < right) [8<9] – TRUE

result.insert(it1+it2, a[mid+it2]) [result[4]=95]

it2 += 1 [it2=3]

while (mid+it2 < right) [9<9] – FALSE

for i in range(0, (it1+it2)) [from 0 to 5-1]

a[left + i] = result[i] [a[5] = 5]

a[left + i] = result[i] [a[5] = 16]

a[left + i] = result[i] [a[6] = 33]

a[left + i] = result[i] [a[7] = 35]

a[left + i] = result[i] [a[8] = 95]

[10, 27, 28, 76, 5, 16, 33, 35, 95]

25. merge(array, left, mid, right) [left=0, mid=4, right=9]

it1 = 0

it2 = 0

while (left+it1 < mid and mid+it2 < right) [0<4 && 4<9] – TRUE

if (a[left+it1] < a[mid+it2]) [10<5] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[0]=5]

it2 += 1 [it2=1]

while (left+it1 < mid and mid+it2 < right) [0<4 && 5<9] – TRUE

if (a[left+it1] < a[mid+it2]) [10<16] – TRUE

result.insert(it1+it2, a[left+it1]) [result[1]=10]

it1 += 1 [it1=1]

while (left+it1 < mid and mid+it2 < right) [1<4 && 5<9] – TRUE

if (a[left+it1] < a[mid+it2]) [27<16] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[2]=16]

it2 += 1 [it2=2]

while (left+it1 < mid and mid+it2 < right) [1<4 && 6<9] – TRUE

if (a[left+it1] < a[mid+it2]) [27<33] – TRUE

result.insert(it1+it2, a[left+it1]) [result[3]=27]

it1 += 1 [it1=2]

while (left+it1 < mid and mid+it2 < right) [2<4 && 6<9] – TRUE

if (a[left+it1] < a[mid+it2]) [28<33] – TRUE

result.insert(it1+it2, a[left+it1]) [result[4]=28]

it1 += 1 [it1=3]

while (left+it1 < mid and mid+it2 < right) [3<4 && 6<9] – TRUE

if (a[left+it1] < a[mid+it2]) [76<33] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[5]=33]

it2 += 1 [it2=3]

while (left+it1 < mid and mid+it2 < right) [3<4 && 7<9] – TRUE

if (a[left+it1] < a[mid+it2]) [76<35] – FALSE

result.insert(it1+it2, a[mid+it2]) [result[6]=35]

it2 += 1 [it2=4]

while (left+it1 < mid and mid+it2 < right) [3<4 && 8<9] – TRUE

if (a[left+it1] < a[mid+it2]) [76<95] – TRUE

result.insert(it1+it2, a[left+it1]) [result[7]=76]

it1 += 1 [it1=4]

while (left+it1 < mid and mid+it2 < right) [4<4 && 8<9] – FALSE

while (left+it1 < mid) [4<4] – FALSE

while (mid+it2 < right) [8<9] – TRUE

result.insert(it1+it2, a[mid+it2]) [result[8]=95]

it2 += 1 [it2=5]

while (mid+it2 < right) [9<9] – FALSE

for i in range(0, (it1+it2)) [from 0 to 9-1]

a[left + i] = result[i] [a[0] = 5]

a[left + i] = result[i] [a[1] = 10]

a[left + i] = result[i] [a[2] = 16]

a[left + i] = result[i] [a[3] = 27]

a[left + i] = result[i] [a[4] = 28]

a[left + i] = result[i] [a[5] = 33]

a[left + i] = result[i] [a[6] = 35]

a[left + i] = result[i] [a[7] = 76]

a[left + i] = result[i] [a[8] = 95]

**Результат роботи: [10, 27, 28, 76, 5, 16, 33, 35, 95]**

1. **Быстрая сортировка**

Ниже представлен псевдокод данного алгоритма:

**array = [28, 76, 27, 10, 5, 35, 95, 16, 33]** # defining an array

**left = 0**

**right = len(array)-1** # so the right = 8 in this case

# main function

**def quickSort(array, left, right) :**

**if (left < right) :**

**r = partition(array, left, right)** # finding middle of array

# calling recursive functions for left and right sides

**quickSort(array, left, r)**

**quickSort(array, r+1, right)**

# function to swap elements

**def swap(a, b) :**

**temp = array[a]**

**array[a] = array[b]**

**array[b] = temp**

# function to divide an array and to position elements

**def partition(array, left, right) :**

**mid = array[(left+right)//2]** # middle of array

**l = left**

**r = right**

**while(l <= r) :**

# finding element in left side that is bigger than middle one

**while(array[l] < mid) :**

**l+=1**

# finding element in right side that is smaller than middle one

**while(array[r] > mid) :**

**r-=1**

# if indexes crossed we must quite the cycle

**if (l >= r) : break**

**swap(l, r)**

**l+=1**

**r-=1**

**return r**

# calling a function

**quickSort(array, left, right)**

**for i in array:**

**print(i, end=' ')**

Пример численного моделирования приведенного псевдокода рекурсивного алгоритма:

Исходные данные array = [28, 76, 27, 10, 5, 35, 95, 16, 33] n=9

left = 0, right = n-1

1. quickSort(array, 0, 8)

if (left < right) - TRUE

partition(array, 0, 8)

mid = array[(0+8)//2] = 5

l = left = 0, r = right = 8

while(0 <= 8) – TRUE

while(array[0] < mid) [28<5] – FALSE

while(array[8] > mid) [33>5] – TRUE

r-=1 [r=7]

while(array[7] > mid) [16>5] – TRUE

r-=1 [r=6]

while(array[6] > mid) [95>5] – TRUE

r-=1 [r=5]

while(array[5] > mid) [35>5] – TRUE

r-=1 [r=4]

while(array[4] > mid) [5>5] – FALSE

if (l >= r) [0>=4] – FALSE

swap(0, 4)

temp = array[0] [= 28]

array[0] = array[4] [= 5]

array[4] = temp

L+=1 [L=1]

r-=1 [r=3]

[5, 76, 27, 10, 28, 35, 95, 16, 33]

while(1 <= 3) – TRUE

while(array[1] < mid) [76<5] – FALSE

while(array[3] > mid) [10>5] – TRUE

r-=1 [r=2]

while(array[2] > mid) [27>5] – TRUE

r-=1 [r=1]

while(array[1] > mid) [76>5] – TRUE

r-=1 [r=0]

while(array[0] > mid) [5>5] – FALSE

if (l >= r) [1>=0] – TRUE

break

[5, 76, 27, 10, 28, 35, 95, 16, 33]

return r [r=0]

r = 0 [ r = partition(array, left, right) ]

2. quickSort(array, left, r) [left=0, right=0]

if (left < right) – FALSE

3. quickSort(array, r+1, right) [r+1=1, right=8]

if (left < right) – TRUE

partition(array, 1, 8)

mid = array[(1+8)//2] = 28

l = left = 1, r = right = 8

while(1 <= 8) – TRUE

while(array[1] < mid) [76<28] – FALSE

while(array[8] > mid) [33>28] – TRUE

r-=1 [r=7]

while(array[7] > mid) [16>28] – FALSE

if (l >= r) [1>=7] – FALSE

swap(1, 7)

temp = array[1] [= 76]

array[0] = array[7] [= 16]

array[4] = temp

L+=1 [L=2]

r-=1 [r=6]

[5, 16, 27, 10, 28, 35, 95, 76, 33]

while(2 <= 6) – TRUE

while(array[2] < mid) [27<28] – TRUE

L+=1 [L=3]

while(array[3] < mid) [10<28] – TRUE

L+=1 [L=4]

while(array[4] < mid) [28<28] – FALSE

while(array[6] > mid) [95>28] – TRUE

r-=1 [r=5]

while(array[5] > mid) [35>28] – TRUE

r-=1 [r=4]

while(array[4] > mid) [28>28] – FALSE

if (l >= r) [4>=4] – TRUE

break

[5, 16, 27, 10, 28, 35, 95, 76, 33]

return r [r=4]

r = 4 [ r = partition(array, left, right) ]

4. quickSort(array, left, r) [left=1, right=4]

if (left < right) – TRUE

partition(array, 1, 4)

mid = array[(1+4)//2] = 27

L = left = 1, r = right = 4

while(1 <= 4) – TRUE

while(array[1] < mid) [16<27] – TRUE

L+=1 [L=2]

while(array[2] < mid) [27<27] – FALSE

while(array[4] > mid) [28>27] – TRUE

r-=1 [r=3]

while(array[3] > mid) [10>27] – FALSE

if (l >= r) [2>=3] – FALSE

swap(2, 3)

temp = array[2] [= 27]

array[2] = array[3] [= 10]

array[3] = temp

L+=1 [L=3]

r-=1 [r=2]

[5, 16, 10, 27, 28, 35, 95, 76, 33]

while(3 <= 2) – FALSE

return r [r=2]

r = 2 [ r = partition(array, left, right) ]

5. quickSort(array, left, r) [left=1, right=2]

if (left < right) – TRUE

partition(array, 1, 2)

mid = array[(1+2)//2] = 16

L = left = 1, r = right = 2

while(1 <= 2) – TRUE

while(array[1] < mid) [16<16] – FALSE

while(array[2] > mid) [10>27] – FALSE

if (l >= r) [1>=2] – FALSE

swap(1, 2)

temp = array[1] [= 16]

array[1] = array[2] [= 10]

array[2] = temp

L+=1 [L=2]

r-=1 [r=1]

[5, 10, 16, 27, 28, 35, 95, 76, 33]

while(2 <= 1) – FALSE

return r [r=1]

r = 1 [ r = partition(array, left, right) ]

6. quickSort(array, left, r) [left=1, right=1]

if (left < right) – FALSE

7. quickSort(array, r+1, right) [r+1=2, right=2]

if (left < right) – FALSE

8. quickSort(array, r+1, right) [r+1=3, right=4]

if (left < right) – TRUE

partition(array, 3, 4)

mid = array[(5+8)//2] = 27

l = left = 3, r = right = 4

while(3 <= 4) – TRUE

while(array[3] < mid) [27<27] – FALSE

while(array[4] < mid) [28>27] – TRUE

r-=1 [r=3]

while(array[3] < mid) [27>27] – FALSE

if (l >= r) [3>=3] – TRUE

break

[5, 10, 16, 27, 28, 35, 95, 76, 33]

return r [r=3]

r = 3 [ r = partition(array, left, right) ]

9. quickSort(array, left, r) [left=3, right=3]

if (left < right) – FALSE

10. quickSort(array, r+1, right) [r+1=4, right=4]

if (left < right) – FALSE

11. quickSort(array, r+1, right) [r+1=5, right=8]

if (left < right) – TRUE

partition(array, 5, 8)

mid = array[(5+8)//2] = 95

l = left = 5, r = right = 8

while(5 <= 8) – TRUE

while(array[5] < mid) [35<95] – TRUE

L+=1 [L=6]

while(array[6] < mid) [95<95] – FALSE

while(array[8] > mid) [33>95] – FALSE

if (l >= r) [6>=8] – FALSE

swap(6, 8)

temp = array[6] [= 95]

array[6] = array[8] [= 33]

array[8] = temp

L+=1 [L=7]

r-=1 [r=7]

[5, 10, 16, 27, 28, 35, 33, 76, 95]

return r [r=7]

r = 7 [ r = partition(array, left, right) ]

12. quickSort(array, left, r) [left=5, right=7]

if (left < right) – TRUE

partition(array, 5, 7)

mid = array[(5+7)//2] = 33

l = left = 5, r = right = 7

while(5 <= 7) – TRUE

while(array[5] < mid) [35<33] – FALSE

while(array[7] > mid) [76>33] – TRUE

r-=1 [r=6]

while(array[6] > mid) [33>33] – FALSE

if (l >= r) [5>=6] – FALSE

swap(5, 6)

temp = array[5] [= 35]

array[5] = array[6] [= 33]

array[6] = temp

L+=1 [L=6]

r-=1 [r=5]

[5, 10, 16, 27, 28, 33, 35, 76, 95]

return r [r=5]

r = 5 [ r = partition(array, left, right) ]

13. quickSort(array, left, r) [left=5, right=5]

if (left < right) – FALSE

14. quickSort(array, r+1, right) [r+1=6, right=7]

if (left < right) – TRUE

partition(array, 6, 7)

mid = array[(6+7)//2] = 35

l = left = 6, r = right = 7

while(6 <= 7) – TRUE

while(array[6] < mid) [35<35] – FALSE

while(array[7] > mid) [76>35] – TRUE

r-=1 [r=6]

while(array[6] > mid) [35>35] – FALSE

if (l >= r) [6>=6] – TRUE

break

[5, 10, 16, 27, 28, 33, 35, 76, 95]

return r [r=6]

r = 6 [ r = partition(array, left, right) ]

15. quickSort(array, left, r) [left=6, right=6]

if (left < right) – FALSE

16. quickSort(array, r+1, right) [r+1=7, right=7]

17. quickSort(array, r+1, right) [r+1=8, right=8]

if (left < right) – FALSE

**Результат роботи: [5, 10, 16, 27, 28, 33, 35, 76, 95]**

1. **Сравнение алгоритмов**

Алгоритм сортировки слиянием: Это рекурсивный алгоритм, который постоянно разбивает список пополам. Если список пуст или состоит из одного элемента, то он отсортирован по определению (базовый случай). Если в списке больше, чем один элемент, мы разбиваем его и рекурсивно вызываем сортировку слиянием для каждой из половин. После того, как обе они уже отсортированы, выполняется основная операция, называемая слиянием. Слияние - это процесс комбинирования двух меньших сортированных списков в один новый, но тоже отсортированный

Алгоритм быстрой сортировки: как и сортировка слиянием, метод основан на подходе "разделяй-и-властвуй". Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

mergeSort используют для упорядочения массивов, лишь если требуется устойчивость метода(которой нет ни у быстрой, ни у пирамидальной сортировок).

Сортировка слиянием является одним из наиболее эффективных методов для односвязных списков и файлов, когда есть лишь последовательный доступ к элементам.

**4) Вывод:**

В работе было рассмотрено два вида логарифмических алгоритмов сортировки: сортировка слиянием и быстрая сортировка(также известна как сортировка Хоара).

Оба рассмотренных алгоритма работают по принципу “разделяй и властвуй”, это означает что задача сортировки массива разбивается на задачи сортировки подмассивов, входящих в его состав. Таким образом удается получить значительный прирост в скорости выполнения сортировки. Главным преимуществом сортировки слиянием является его устойчивость, но необходимо относительно много памяти. Быстрая же сортировка требует мало памяти и может выполняться быстрее других алгоритмов, но скорость выполнения может легко деградировать до O(n^2), поэтому часто используется с некоторыми доработками и является одним из наиболее широко используемых методов сортировки. В моём случае алгоритм быстрой сортировки сработал значительно быстрее чем сортировка слиянием.