МАИ

Лабораторная работа №11

 «Методы поиска приближённого значения трансцендентного уравнения и их программная реализация на основе итерационного и рекурсивного подходов»Вариант №14

Факультет робототехнических и интеллектуальных систем

Кафедра «Системы приводов летательных аппаратов»

Выполнил:

Студент группы М7О-114БВ-24

Фельдман Лев Борисович

Проверил:  
Доцент Кафедры 702 Козлова Н.М.

Ассистент Кафедры 702 Милославский Я.Г.

Москва 2025

**Цель работы: знакомство с пирамидальным методом сортировки, методом быстрой сортировки и сортировкой методом слияния.**

**Рекурсивный алгоритм** — это алгоритм, который решает задачу путём многократного (вложенного) вызова самого себя для решения более простых (подзадач), пока не будет достигнуто некоторое базовое условие (условие выхода), при котором дальнейшие вызовы прекращаются.

Основные компоненты рекурсивного алгоритма

1. Базовое условие (условие выхода)  
   Определяет, при каком входе рекурсия завершается без новых вызовов. Без корректно заданного базового условия рекурсия «уйдёт в бесконечность» и вызовет переполнение стека.
2. Рекурсивный вызов  
   Часть алгоритма, где функция вызывает сама себя с новым (обычно упрощённым) аргументом.

**Пример 1. Факториал**

Факториал числа n определяется как:

n!=

function f = factorial\_rec(n)

% Рекурсивная функция для вычисления факториала n

if n == 0 f = 1; % базовый случай

else f = n \* factorial\_rec(n-1); end % рекурсивный шаг

end

При factorial(3) ход вычислений будет таким:  
3 \* factorial(2) → 3 \* (2 \* factorial(1)) → 3 \* (2 \* (1 \* factorial(0))) → 3 \* 2 \* 1 \* 1 → **6**.

**Пример 2. Числа Фибоначчи**

Последовательность Фибоначчи:

F

function f = fib\_rec(n)

% Рекурсивная функция для n-го числа Фибоначчи

if n == 0 f = 0; % базовый случай 1

elseif n == 1 f = 1; % базовый случай 2

else f = fib\_rec(n-1) + fib\_rec(n-2); % рекурсивный шаг

end

end

>>fib\_rec(10)

ans =

55

**Пример 3. Быстрый подсчёт суммы элементов в массиве**

Допустим, у нас есть вектор AAA, и мы хотим рекурсивно получить сумму всех его элементов:

function s = sum\_rec(A)

% Рекурсивная функция для суммы элементов вектора A

if isempty(A) s = 0; % базовый случай: пустой массив

else s = A(1) + sum\_rec(A(2:end)); % рекурсивный шаг

end

end

>>sum\_rec([4, 7, 1, 3])

ans =

15

## Пирамидальная сортировка

function B = heapsort(B)

% HEAPSORT Сортировка массива-матрицы B по первому столбцу методом "пирамидальной сортировки"

n = size(B,1);

% 1) Построение мин-кучи: вставляем по одному элементу

for i = 2 : n

B = heapifyup1(B, i);

end

% 2) Извлечение элементов из кучи в конец массива

for l = n : -1 : 2

% Меняем корень (минимум) с последним элементом текущей кучи

B([1 l], :) = B([l 1], :);

% Восстанавливаем мин-кучу на префиксе [1..l-1]

B = heapifydownN1(B, 1, l-1);

end

function B = heapifydownN1(B, i, l)

% HEAPIFYDOWNN1 Просеивание вниз для мин-кучи

left = 2 \* i;

if left > l

return;

end

right = left + 1;

smallest = i;

if left <= l && B(left,1) < B(smallest,1)

smallest = left;

end

if right <= l && B(right,1) < B(smallest,1)

smallest = right;

end

if smallest ~= i

B([i smallest], :) = B([smallest i], :);

B = heapifydownN1(B, smallest, l);

end

end

function B = heapifyup1(B, i)

% HEAPIFYUP1 Просеивание вверх для мин-кучи

if i == 1

return;

end

parent = floor(i/2);

if B(i,1) < B(parent,1)

B([i parent], :) = B([parent i], :);

B = heapifyup1(B, parent);

end

end

end