Т1Ч9

Часть 9. Анализ алгоритмов.

Задание 9.2

1. Дан алгоритм:

Алгоритм Secret(A)
// Входные данные: массив вещественных чисел A $[0...n-1]$
$M \leftarrow A[0]$
for $i \in [0...n-1]$ **do**
  for $j \in [i...n-1]$ **do**
    $S \leftarrow 0$
    for $k \in [i...j]$ **do**
      $S \leftarrow S + A[k]$
    if $S > M$
      $M \leftarrow S$
return M

а) Алгоритм находит максимально возможную сумму элементов подмножеств из множества A и возвращает это число.
б) Основная операция - сложение
в) $C(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j-i+1) = \sum_{i=0}^{n-1} \frac{(n-i+1)(n-i)}{2} = \frac{n(n+1)(n+2)}{6}$
г) Эффективность - $O(n^3)$
д) Существует алгоритм Кадане, который выполняет ту же операцию с эффективностью - $O(n)$

```
Функция Кадане(A):
    Вход: массив A[0..n-1] вещественных чисел
    Выход: максимальная сумма непрерывного подмассива

    max_current ← A[0]  // текущая максимальная сумма
    max_global ← A[0]   // глобальная максимальная сумма

    Для i от 1 до n-1 выполнить:
        // Выбираем максимум между текущим элементом и суммой предыдущих
        max_current ← максимум(A[i], max_current + A[i])

        // Обновляем глобальный максимум при необходимости
        Если max_current > max_global:
            max_global ← max_current

    Вернуть max_global
```

Задание 9.4

1. Рекурсивный алгоритм вычисления (2^n)

а) Разработка алгоритма:
Рекурсивный алгоритм на основе формулы ($2^n = 2^{n-1} + 2^{n-1}$):

```
Алгоритм PowerOfTwo(n)
    Входные данные: целое неотрицательное число n
    Выходные данные: значение 2^n

    если n == 0
        вернуть 1
    иначе
        вернуть PowerOfTwo(n-1) + PowerOfTwo(n-1)
```

б) Рекуррентное уравнение:
Основная операция — сложение.
Количество операций (C(n)):

$$C(n) = 2 \cdot C(n-1) + 1 \quad \text{при} \quad C(0) = 0$$

Решение методом обратной подстановки:

$$C(n) = 2^n - 1 \in \Theta(2^n)$$

в) Дерево рекурсивных вызовов:

- Каждый вызов с аргументом (n) порождает 2 вызова с (n-1).

- Общее количество вызовов: $(2^{n+1} - 1)$ (экспоненциальный рост).

г) Оценка алгоритма:
Алгоритм крайне неэффективен из-за экспоненциального времени работы. Лучше использовать итеративный метод (умножение) или возведение в степень через квадраты $(O(\log n))$.

---

2. Алгоритм `Min1`

а) Что вычисляет:
Находит минимальный элемент в массиве (A).

б) Основная операция:
Сравнение элементов $(m \leq A[n-1])$.

в) Рекуррентное уравнение:

$$C(n) = C(n-1) + 1 \quad \text{при} \quad C(1) = 0$$

Решение:

$$C(n) = n - 1 \in \Theta(n)$$

---

3. Алгоритм `Min2`

а) Рекуррентное уравнение:
Основная операция — сравнение $(a \leq b)$.
Для массива длины $(n = 2^k)$:

$$C(n) = 2 \cdot C(n/2) + 1 \quad \text{при} \quad C(1) = 0$$

Решение:

$$C(n) = n - 1 \in \Theta(n)$$

(Аналогично линейному проходу, но с дополнительными накладными расходами на рекурсию.)

Задание 9.6

Определение класса эффективности по времени выполнения:

Размер данных ((n))	Время выполнения (ms)
1000	12
2000	15
3000	18
4000	22
5000	25
6000	27
7000	29
8000	30

Анализ:

- Время растет примерно линейно с увеличением (n).
- Отношение ($\frac{T(n)}{n}$) стремится к константе (примерно (0.00375) ms/элемент).

Вывод:
Алгоритм имеет **линейную сложность** (O(n)).