Algorithms and data structures

lecture #3. Recursion, Stack

Mentor: Rustam Khakov

lecture #3. Recursion, Stack

- Recursion, Stack
  - Что такое рекурсия
  - Математическая интерпретация
  - Как хранится в памяти
  - Базовое условие в рекурсии
  - Хвостовая и нехвостовая рекурсия
  - Выделение памяти для разных вызовов
  - Рекурсия VS Итерация
  - Недостатки рекурсивного по сравнению с итеративным программированием
  - Итоги и резюме по рекурсии
  - Stack как структуда данных

Простые сортировки

Название	Лучшее время	Среднее	Худшее	Память	Устойчивость	Обмены (в среднем)	Описание
Сортировка пузырьком (Bubble Sort)	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Да	$O(n^2)$	Алгоритм состоит в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами.
Сортировка вставками (Insertion Sort)	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Да	$O(n^2)$	На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива до тех пор, пока весь набор входных данных не будет отсортирован.
Сортировка выбором (Selection Sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Нет	$O(n)$	На i-ом шаге алгоритма находим минимальный среди последних $n - i + 1$, и меняем его местами с i-ым элементом в массиве.

3.

4. Сортировка выбором

```
function selectionSort(T[n] a):
   for i = 0 to n - 2
     min = i
     for j = i + 1 to n - 1
       if a[j] < a[min]
         min = j
     swap(a[i], a[min])
```

5. Вставками

```
for i = 2 to n do
    x = A[i]
    j = i
    while (j > 1 and A[j-1] > x) do
        A[j] = A[j-1]
        j = j - 1
    end while
    A[j] = x
end for[6]
```

Простые сортировки

Шейкерная

Сложность по времени

Худшее время: $O(n^2)$
Среднее время: $O(n^2)$
Лучшее время: $O(n)$

Затраты памяти: $O(1)$

Расческой

Сложность по времени

Худшее время: $O(n^2)$
Среднее время: $\Omega(n^2/2^p)$, где p — количество инкрементов
Лучшее время: $O(n \log n)$

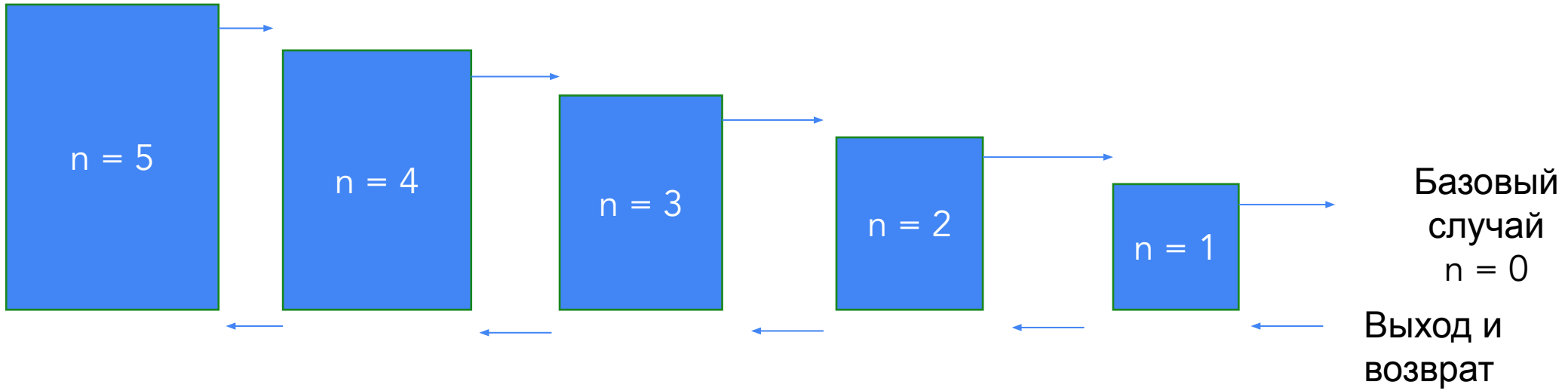
Затраты памяти: $O(1)$

What is Recursion

Процесс, в котором функция прямо или косвенно вызывает сама себя, называется рекурсией, а соответствующая функция называется рекурсивной функцией.

Важно! Мы должны обеспечить определенный случай, чтобы завершить этот процесс рекурсии.

Каждый раз функция вызывает себя с более простой версией исходной задачи.



Задача сводится к конкретному значению

Recursion – математическая интерпретация

Подход №1
n = 5
function(n) = for(1+2+3+4+…+n) -> res = res+1, i<=n

Подход №2
n=5
function(n) = n+function(n-1) -> n = 1

Memory and Stack

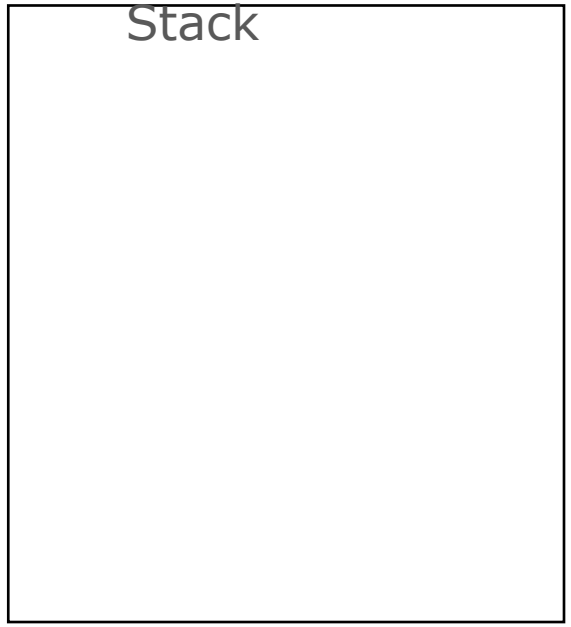
- Рекурсия использует больше памяти
- Рекурсивная функция использует структуру LIFO (Last In First Out)

- Память для вызываемой функции выделяется поверх памяти, выделенной для вызывающей функции.
- Для каждого вызова функции создается другая копия локальных переменных.
- Когда бызовый случай достигнут, функция возвращает свое значение функции, которой она вызывается, и память освобождается.

Стек – линейная структура данных, которая следует определенному порядку выполнения операций.
Четыре основные операции:

1. push
2. pop
3. isEmpty
4. peek

Пример Stack



Стек(The Stack)

- FILO (First in last out) - первый зашел, последний вышел

- выполнение методов

Рекурсия

Для того чтобы понять рекурсию, надо сначала понять рекурсию

Рекурсия - вызов самого себя

Базовое условие

Рекурсия работает пока не достигнет базового случая

```
int fanc(int n) {
If (n<=1) // base case
      return 1;
else
      return n*fact(n-1);
}
```

```
int fanc(int n) {
If (n==100) // base case
      return 1;
else
      return n*fact(n-1);
}
```

Вопрос ???
Какой базовый случай сработает, если n = 30.

Типы рекурсии

Прямая рекурсия – если функция вызывает ту же функцию.
Косвенная рекурсия – если функция вызывает другую функцию, а другая функция прямо или косвенно вызывает первую.

```
int directRec() {
// … some code
     directRec()
// … some code
}
```

```
int indirectRec1() {
// … some code
     indirectRec2()
// … some code
}

int indirectRec2() {
// … some code
     indirectRec1()
// … some code
}
```

Recursion VS Iteration

Рекурсия	Итерация
Прекращается, когда базовый случай становиться истинным	Прекращается когда условие становиться ложным
Используется с функциями	Используется с циклами
Каждому рекурсивному вызову требуется дополнительное место в памяти стека	Каждая итерация не требует дополнительного места
Меньший размер кода	Больший размер кода

Рекурсивные и итерационные подходы обладают одинаковыми возможностями для решения задач.
Недостатки?
Преимущества?

Итог

- В рекурсии есть два типа случаев: рекурсивные случай и базовый
- Базовый случай используется для завершения рекурсивной функции
- Каждый рекурсивняй вызов создает новую копию этого метода в памяти стека
- Бесконечная рекурсия может привести к нехватке памяти (StackOverFlow)
- Примеры: сортировка слиянием, быстрая сортировка, Ханойская башня, ряд Фибонначи, Факториальная задача, ...