План на занятие:

1. Алгоритмы
    1.1. Модель вычислений
    1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
    1.3. Асимптотические обозначения
    1.4. Скорость роста и отношения доминирования
    2.5. Сортировка и бинарный поиск
    2.6. Подходы к построению алгоритма

План на занятие:

1. Алгоритмы
    1.1. Модель вычислений
    1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
    1.3. Асимптотические обозначения
    1.4. Скорость роста и отношения доминирования
    2.5. Сортировка и бинарный поиск
    2.6. Подходы к построению алгоритма

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг
- Подпрограммы и циклы состоят из нескольких простых операций

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг
- Подпрограммы и циклы состоят из нескольких простых операций

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг
- Подпрограммы и циклы состоят из нескольких простых операций
- Каждое обращение к памяти занимает один временной шаг, компьютер обладает неограниченным объемом оперативной памяти

Модель вычислений

Модель вычислений RAM:

- подход к определению сложности операций на гипотетическом компьютере (random access machine).

Цель этого моделирования заключается в том, чтобы определить время исполнения программы (алгоритма) по операциям (шагам) в данном конкретном алгоритме.

Принцип работы:

- "Простая" операция (+, *, -, =, if, вызов функции, создание переменной) требует ровно один временный шаг
- Подпрограммы и циклы состоят из нескольких простых операций
- Каждое обращение к памяти занимает один временной шаг, компьютер обладает неограниченным объемом оперативной памяти (забили на память)

План на занятие:

1. Алгоритмы
   1.1. Модель вычислений
   1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
   1.3. Асимптотические обозначения
   1.4. Скорость роста и отношения доминирования
   2.5. Сортировка и бинарный поиск
   2.6. Подходы к построению алгоритма

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Рассмотрим для начала простой алгоритм поиска значения в массиве:

```
value_to_find = 42
```

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Рассмотрим для начала простой алгоритм поиска значения в массиве:

```
value_to_find = 42
for item in array:
    if item == value_to_find:
        print(item)
        break
```

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Рассмотрим для начала простой алгоритм поиска значения в массиве:

```
value_to_find = 42
for item in array:  # объявление цикла (состоит из нескольких операций)
    if item == value_to_find:  # простая операция 'if'
        print(item)  # простая операция вызов функции
        break  # остановка цикла
```

Анализ сложности

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Рассмотрим для начала простой алгоритм поиска значения в массиве:

```python
value_to_find = 42
for item in array:  # объявление цикла (состоит из нескольких операций)
    if item == value_to_find:  # простая операция 'if'
        print(item)  # простая операция вызов функции
        break  # остановка цикла
```

Наилучший случай: array.index(value_to_find) == 0

Анализ сложности

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Рассмотрим для начала простой алгоритм поиска значения в массиве:

```
value_to_find = 42
for item in array:  # объявление цикла (состоит из нескольких операций)
    if item == value_to_find:  # простая операция 'if'
        print(item)  # простая операция вызов функции
        break  # остановка цикла
```

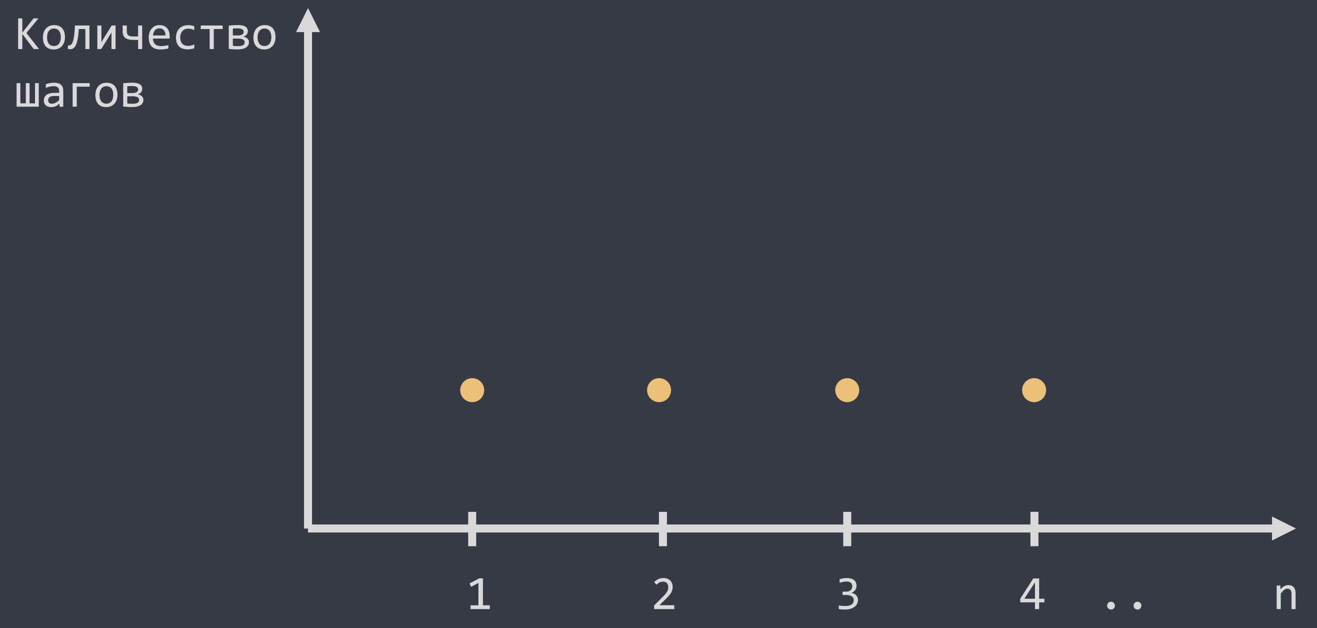
```
Наилучший случай: array.index(value_to_find) == 0
Наихудший случай: array.index(value_to_find) == len(array)  # пусть len(array) == n
```

Анализ сложности

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

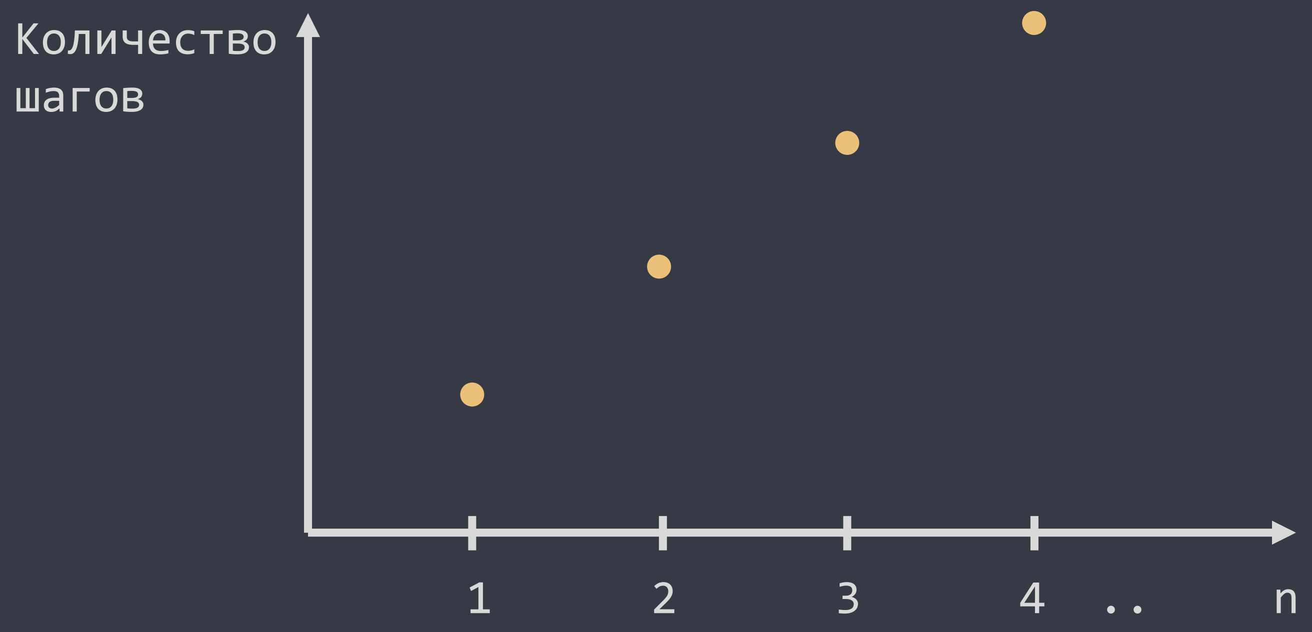
```
Наилучший случай: array.index(value_to_find) == 0
Наихудший случай: array.index(value_to_find) == len(array)  # пусть len(array) == n
```



Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

```
Наилучший случай: array.index(value_to_find) == 0
Наихудший случай: array.index(value_to_find) == len(array)  # пусть len(array) == n
```

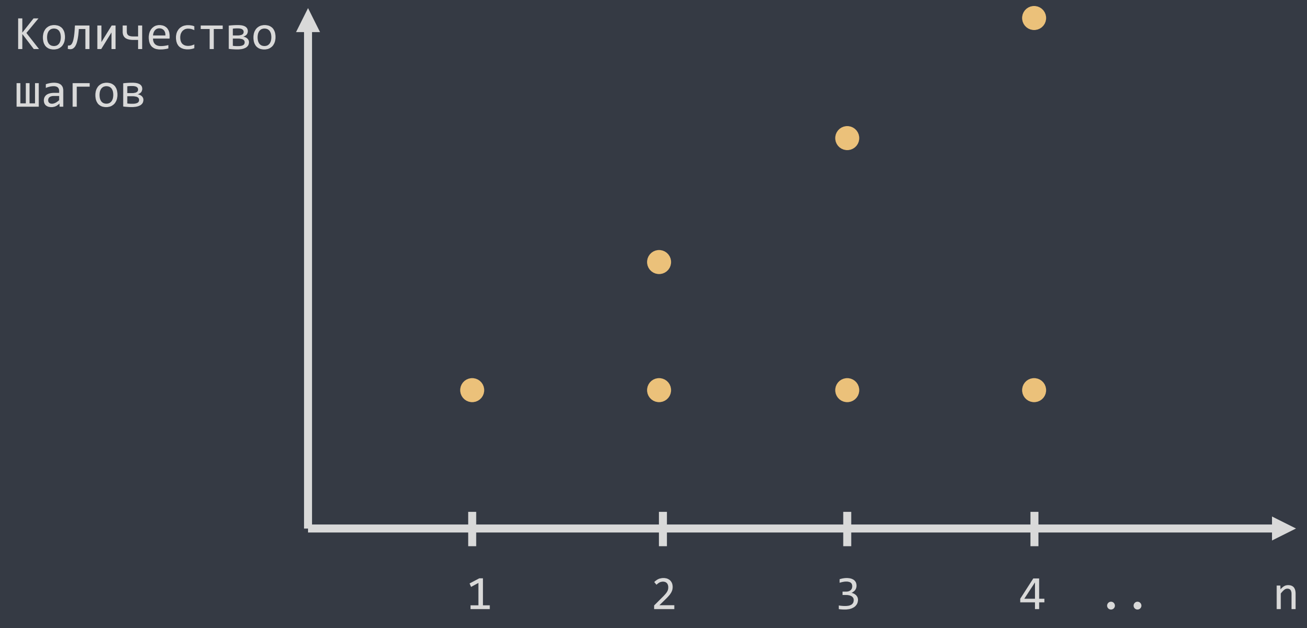


Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Наилучший случай: array.index(value_to_find) == 0

Наихудший случай: array.index(value_to_find) == len(array) # пусть len(array) == n

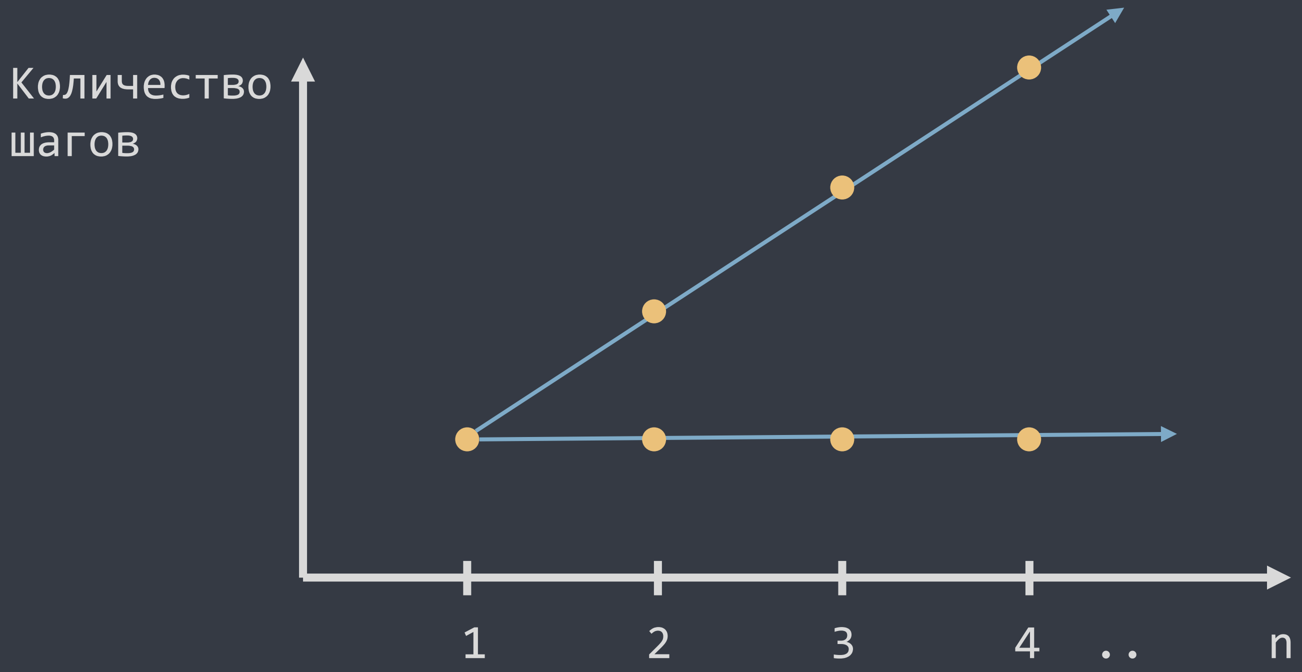


Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Наилучший случай: `array.index(value_to_find) == 0`

Наихудший случай: `array.index(value_to_find) == len(array)  # пусть len(array) == n`



Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Сложность алгоритма в наилучшем случае – функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n

Анализ сложности

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

Сложность алгоритма в наилучшем случае – функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n

Сложность алгоритма в наихудшем случае – функция, определяемая максимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n

Анализ сложности

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных

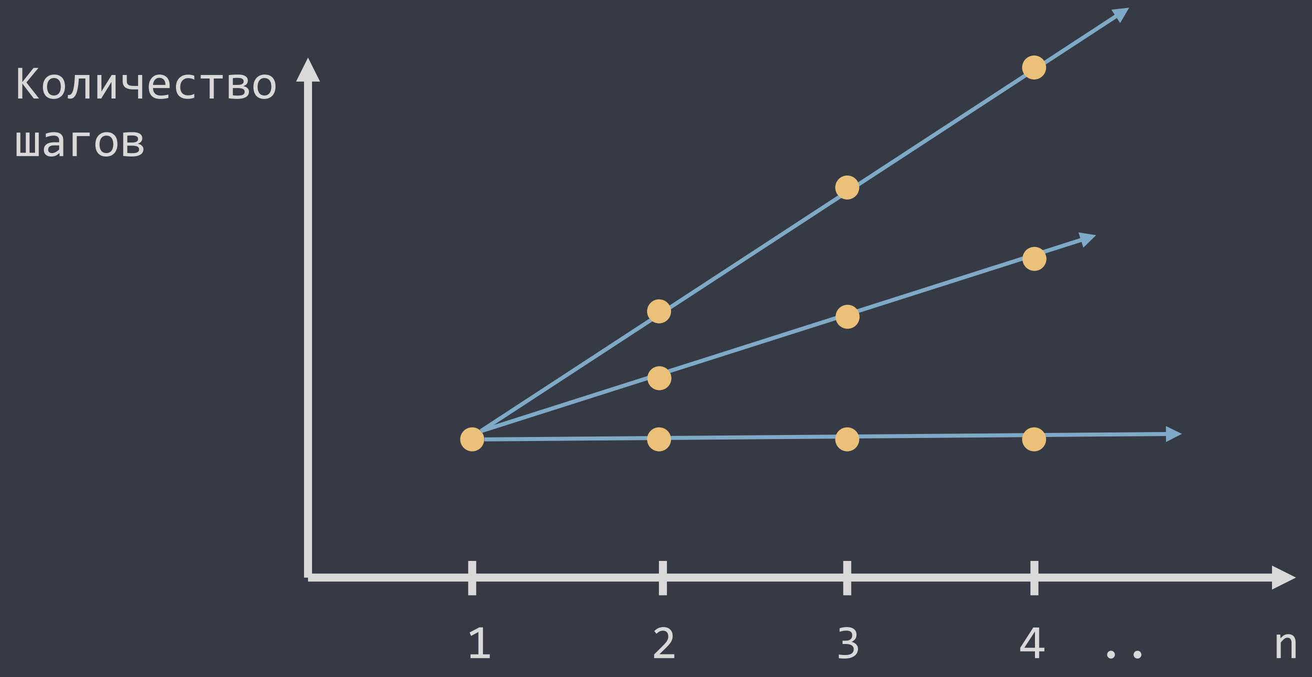
Сложность алгоритма в наилучшем случае – функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n

Сложность алгоритма в наихудшем случае – функция, определяемая максимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n

Сложность алгоритма в среднем случае – --//-- определяемая средним количеством шагов

Наилучший и наихудший случаи:

- время выполнения алгоритма (в единичных временных шагах) при разных входных данных



План на занятие:

1. Алгоритмы
   - 1.1. Модель вычислений
   - 1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
   - 1.3. Асимптотические обозначения
   - 1.4. Скорость роста и отношения доминирования
   - 2.5. Сортировка и бинарный поиск
   - 2.6. Подходы к построению алгоритма

Пример:

Допустим, мы высчитали сложность нашего алгоритма в наихудшем случае для множества n и получили функцию, которая описывается следующим уравнением:

$$T(n) = 123574n^2 + 4353n + 8341\, lg_2(n) + 13564$$

Пример:

Допустим, мы высчитали сложность нашего алгоритма в наихудшем случае для множества n и получили функцию, которая описывается следующим уравнением:

$$T(n) = 123574n^2 + 4353n + 8341\, lg_2(n) + 13564$$

Проблемы:

1. Сама оценка и получение точной математической функции уже является проблемой при наличии лишь точек на графике (требуется аппроксимация)

Пример:

Допустим, мы высчитали сложность нашего алгоритма в наихудшем случае для множества n и получили функцию, которая описывается следующим уравнением:

$$T(n) = 123574n^2 + 4353n + 8341\,lg_2(n) + 13564$$

Проблемы:

1. Сама оценка и получение точной математической функции уже является проблемой при наличии лишь точек на графике (требуется аппроксимация)
2. Эта функция не особо информативная, в ней содержится много лишней информации, кроме той, что с увеличением n временная сложность возрастает квадратически

Асимптотические обозначения

Пример:

Допустим, мы высчитали сложность нашего алгоритма в наихудшем случае для множества n и получили функцию, которая описывается следующим уравнением:

$$T(n) = 123574n^2 + 4353n + 8341\, lg_2(n) + 13564$$

Проблемы:

1. Сама оценка и получение точной математической функции уже является проблемой при наличии лишь точек на графике (требуется аппроксимация)
2. Эта функция не особо информативная, в ней содержится много лишней информации, кроме той, что с увеличением n временная сложность возрастает квадратически

=> более информативными будут асимптотические величины $(n \to +\infty)$

Асимптотические обозначения

Формальные определения:

- $f(n) = O(g(n))$ - функция $f(n)$ ограничена сверху функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \leq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = O(n^2)$

- $f(n) = \Omega(g(n))$ - функция $f(n)$ ограничена снизу функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \geq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = \Omega(n)$

Асимптотические обозначения

Формальные определения:

- $f(n) = O(g(n))$ - функция $f(n)$ ограничена сверху функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \leq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = O(n^2)$

- $f(n) = \Omega(g(n))$ - функция $f(n)$ ограничена снизу функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \geq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = \Omega(n)$

Верно ли?

- $T(n) = 12n^2 + 12n = O(n^3)$
- $T(n) = 12n^2 + 12n = \Omega(n^2)$
- $T(n) = 12n^2 + 12n = \Omega(n^3)$

Асимптотические обозначения

Формальные определения:

- $f(n) = O\big(g(n)\big)$ - функция $f(n)$ ограничена сверху функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \leq c * g(n)\ \forall n$
    - Пример $T(n) = 12n^2 + 12n = O(n^2)$

- $f(n) = \Omega\big(g(n)\big)$ - функция $f(n)$ ограничена снизу функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \geq c * g(n)\ \forall n$
    - Пример $T(n) = 12n^2 + 12n = \Omega(n)$

Верно ли?

- $T(n) = 12n^2 + 12n = O(n^3)$
- $T(n) = 12n^2 + 12n = \Omega(n^2)$
- $T(n) = 12n^2 + 12n = \Omega(n^3)$

Асимптотические обозначения

Формальные определения:

- $f(n) = O(g(n))$ - функция $f(n)$ ограничена сверху функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \leq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = O(n^2)$

- $f(n) = \Omega(g(n))$ - функция $f(n)$ ограничена снизу функцией $c * g(n)$, где c – произвольная константа. Т.е. существует такая c, что при $n \geq n_0$ выполняется $f(n) \geq c * g(n)\ \forall n$
  - Пример $T(n) = 12n^2 + 12n = \Omega(n)$

Верно ли?

- $T(n) = 12n^2 + 12n = O(n^3)$ выбрана функция $12 * n^3$
- $T(n) = 12n^2 + 12n = \Omega(n^2)$ выбрана функция $10 * n^2$
- $T(n) = 12n^2 + 12n = \Omega(n^3)$

План на занятие:

1. Алгоритмы
   1.1. Модель вычислений
   1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
   1.3. Асимптотические обозначения
   1.4. Скорость роста и отношения доминирования
   2.5. Сортировка и бинарный поиск
   2.6. Подходы к построению алгоритма

Отношения доминирования

Доминирование:

Одна функция - f доминирует над другой - g (менее быстро растущей), когда $f(n) = O\big(g(n)\big)$

Записывается так: $g >> f$

Отношения доминирования

Доминирование:

Одна функция - f доминирует над другой - g (менее быстро растущей), когда $f(n) = O\big(g(n)\big)$

Записывается так: $g >> f$

Отношения доминирования самых популярных классов функций:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n * \log(n) \gg n \gg \log(n) \gg 1$$

План на занятие:

1. Алгоритмы
    1.1. Модель вычислений
    1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
    1.3. Асимптотические обозначения
    1.4. Скорость роста и отношения доминирования
    2.5. Сортировка и бинарный поиск
    2.6. Подходы к построению алгоритма

Вложенный цикл:

Речь идет не о каком-либо конкретном алгоритме, а просто о цикле внутри цикла. Важно понимать и привыкнуть к тому, что если вы видите цикл внутри цикла, то это автоматически означает асимптотическую сложность наихудшего случая не меньше, чем $O(n^2)$

Вложенный цикл:

Речь идет не о каком-либо конкретном алгоритме, а просто о цикле внутри цикла. Важно понимать и привыкнуть к тому, что если вы видите цикл внутри цикла, то это автоматически означает асимптотическую сложность наихудшего случая не меньше, чем $O(n^2)$

```
for row in table:
    for value in row:
        ...
```

Вложенный цикл:

Речь идет не о каком-либо конкретном алгоритме, а просто о цикле внутри цикла. Важно понимать и привыкнуть к тому, что если вы видите цикл внутри цикла, то это автоматически означает асимптотическую сложность наихудшего случая не меньше, чем $O(n^2)$

```
for row in table:
    for value in row:
        ...
```

Часто вложенных циклов можно избежать, предварительно отсортировав данные или использовав другую структуру данных (к примеру, словарь). А, может, и еще проще, просто добавив несколько счетчиков (переменные, которые мы изменяем на каждом шаге цикла для того, чтобы следить за каким-либо состоянием или показателем).

Популярные алгоритмы и их сложность

Сортировка:

Алгоритмы сортировки данных в массиве

Сложность оптимальных алгоритмов сортировки = $n * \log(n)$

Такие алгоритмы как "сортировка слиянием"(merge sort), "быстрая сортировка"(quick sort) и многие другие: https://habr.com/ru/post/335920/

Некоторые алгоритмы используют когда важна не асимптотическая сложность наихудшего случая, а, к примеру, средняя сложность для определенного n. Т.е. уже подгоняют под определенную задачу

Популярные алгоритмы и их сложность

Бинарный поиск:

Алгоритм поиска данных в отсортированном массиве

Сложность = $\log(n)$

Представляет из себя важный принцип подхода к задачам алгоритмов. Суть алгоритма в том, чтобы взять центральный элемент из отсортированного списка и сравнить его со значением поиска. Таким образом мы наполовину сократим наш массив поиска. Так делается до тех пор, пока значение не будет найдено. В наихудшем случае мы будем делить массив пополам до того момента, пока не останется 1 элемент => максимальное количество итераций = $\log(n)$

План на занятие:

1. Алгоритмы
    1.1. Модель вычислений
    1.2. Анализ сложности наилучшего, наихудшего и среднего случаев
    1.3. Асимптотические обозначения
    1.4. Скорость роста и отношения доминирования
    2.5. Сортировка и бинарный поиск
    2.6. Подходы к построению алгоритма

Популярные алгоритмы и их сложность

1. Попробуйте отсортировать данный вам массив. После сортировки сложность многих задач снижается и асимптотическая сложность алгоритма может уменьшиться.

2. Попробуйте использовать другую структуру данных. Вообще говоря, перевод из одной структуры данных в другую может занимать довольно много времени и важно представлять асимптотическую оценку этого перевода. Но некоторые задачи решаются намного быстрее со словарями или другими структурами данных (деревья, очереди), чем со списками (массивами).

3. Попробуйте разбить задачу на подзадачи (как в бинарном поиске). Вообще эта тема больше связана с рекуррентными функциями, что мы не проходили, но это тоже важный шаг, который стоит рассмотреть при построении оптимального алгоритма.