**BigO BigTheta BigOmega**

За n→inf

1. O(F) - определя множеството от всички функции f, които нарастват не по-бързо от F, т.е. съществува константа c > 0 такава, че f (n) <= cF(n).

2. θ(F) - определя множеството от всички функции f, които нарастват толкова бързо, колкото и F (с точност до константен множител), т.е. съществуват константи c1 >0 и c2 > 0 такава, че c1 F(n) <= f (n) <= c2 F(n).

3. Ω(F) - определя множеството от всички функции f, които нарастват не по-бавно от F, т.е. съществува константа c > 0 такава, че f (n) >= cF(n).

**O(F), свойства**

* Елементарна операция (не зависи от размера на обработваните данни) - O(1)
* Рефлексивност: f ∈ О( f );
* Транзитивност: ако f ∈ О(g), g ∈ О(h), то f ∈ О(h);
* Транспонирана симетрия: ако f ∈ Ω(g), то g ∈ O( f ) и обратно;
* За всяко k > 0, kf ∈ О(f);
* n^r ∈О(n^s), за 0 < r < s.
* Нарастването на сума от функции: f + g ∈ max(O( f ), O(g));
* Композиция на оператори - f \*g ∈ O(f\*g);
* Условни оператори - определя се от асимтотично най-бавния между условието и различните случаи;
* Цикли, вложени цикли - O(n), O(n^p)

**Недостатъци на асимптотична нотация**

* Оценен алгоритъм да е по-бавен от алгоритъм с по-лоша оценка
* Два алгоритъма с еднаква сложност може да са различно бързи.

**Най-често използвани функции, подредени по скорост на нарастване**

c, log n, n, n.log n, n^2, n^3, 2^n, n!, n^n

**Stack**

Хомогенна линейна структура от данни. - „последен влязъл - пръв излязъл“ (LIFO)

Операции: empty() push(x) pop() top()

Последователно представяне: Масив – края (последния добавен) е top

Свързано представяне: Списък – top сочи към първия елемент

**Queue**

Хомогенна линейна структура от данни - „първи влязъл - пръв излязъл“ (FIFO)

Операции: empty() push(x) pop() head/front()

Последователно представяне: Масив началото на който е head краят е tail

Свързано представяне: Списък – tail e най-вдясно и е последния добавен елемент head е най-вляво и е първия добавен елемент.

Циклична Опашка – tail сочи към front

**Homogeneous data**: Homogeneous data structures are those data structures that contain only similar type of data e.g. like a data structure containing only integer or float values. The simplest example of such type of data structures is an Array.

**Heterogeneous Data**: Heterogeneous Data Structures are those data structures that contains a variety or dissimilar type of data, for e.g. a data structure that can contain various data of different data types like integer, float and character. The examples of such data structures include structures, union etc.

**Двоично дърво**

Двоично дърво от тип T е рекурсивна структура от данни, която е или празна или е образувана от:

* Данна от тип T, наречена корен на двоичното дърво;
* Двоично дърво от тип T, наречено ляво поддърво на двоичното дърво;
* Двоично дърво от тип Т, наречено дясно поддърво на двоичното дърво.

**Множеството на върховете (възлите) на едно двоично дърво се определя рекурсивно:**

* Празното двоично дърво няма върхове.
* Върховете на непразно дърво са неговият корен и върховете на двете му поддървета.

**Елементи на дърво**

* Листа - върховете с две празни поддървета.
* Вътрешни върхове - върховете, различни от корена и листата.
* Ляв наследник на един връх - коренът на лявото му поддърво (ако то е непразно).
* Десен наследник на един връх - коренът на дясното му поддърво (ако то е непразно).
* Ако a е наследник на b (ляв или десен), казваме, че b е родител (баща) на a.

Predecessor

Successor

**Над структурата от данни двоично дърво са възможни следните операции:**

* Достъп до връх – възможен е пряк достъп до корена и непряк достъп до останалите върхове.
* Създаване и изтриване на дърво – възможни са добавяне и премахване на върхове на произволно място в двоичното дърво, но резултатът трябва отново да е двоично дърво от същия тип.
* Обхождане – това е метод, позволяващ да се осъществи достъп до всеки връх на дървото един единствен път.

**Обхождането** е рекурсивна процедура, която се осъществява чрез изпълнение на следните три действия, в някакъв фиксиран ред:

* Обхождане на корена.
* Обхождане на лявото поддърво.
* Обхождане на дясното поддърво.

**Представяне на двоично дърво:**

* Свързано
* Верижно
* Чрез списък на бащите

**Идеално балансирано двоично наредено дърво**

* Броят на възлите в лявото и дясното поддърво се различава най-много с 1.
* Лявото и дясното поддървета са идеално балансирани двоично наредени дървета.

**Балансирано двоично наредено дърво**

* Височините на лявото и дясното поддърво се различават най-много с 1.
* Лявото и дясното поддървета са балансирани двоично наредени дървета.
* Всяко идеално балансирано дърво е балансирано дърво, но обратното не е вярно.

**Алгоритъм за създаване на идеално балансирано дърво**

* Eлементите, които ще се включват към празното двоично наредено дърво се подават в нарастващ ред.
* Предварително е известен броят на върховете на дървото.

n - брой на върховете в дървото

n = nLeft + nRight + 1 && |nLeft-nRight| <= 1