자료구조의 필요성을 언급하기 이전에 자료구조의 정의를 먼저 확인할 필요가 있다. '자료'는 컴퓨터가다루는 자원을 뜻하며 그 말인 즉슨 인간이 분석하거나 얻기를 원하는 요소라는 뜻이다. 쉽게 말하면 컴퓨터가 다루는 최소 단위인 '데이터' 라 고생각하면 된다. 인간이 그 자료를 가지고 어떠한 결과물을 얻기 위해 컴퓨터의도움이 필요한데 문제는 인간과 컴퓨터는 근본적으로 알아듣는 언어가 다르다. 그러다보니 수월하게서 로소통을 하기 위해 형성된 '구조'가 자료구조인 것이다.(물론 그 이전에 서로 소통하기 위한 '프로그래밍언어'가 인간과 컴퓨터를 잇는 매개체 이지만 그 수단인 자료를 어떻게 구성하느냐가 더욱 중요하게작용한다.) 인간도 직관적으로 알아볼 수 있고, 컴퓨터도 효율적으로 자원을 저장할 수 있게끔 구성되었으며 한정된 컴퓨터의 자원(메모리)를 절약하고 효율적으로 사용할 수 있게 설계된 것이 자료구조이다. 자료구조 덕분에 인간은 빠른 컴퓨터를 더 빠르고 효율적으로 사용할 수 있게 되었다. 흔히들 자료구조를 표현할 때 '어떤 물건을 창고에어떻게 쌓을지에 대한 방법' 이라고 표한하는데 이는 다소 좁은 시각의 표현이라고 생각한다. 창고에 그저 물건을 쌓기위한 방법이라면 '최선의 방법' 이라는게 존재할 것이다. 하지만 컴퓨터의 경우에는 창고의 형태가바뀔 수도 있고, 물건의 형태가 바뀔수도 있으며, 그 창고의 출입문 위치가 바뀔수도 있다. 이런 수많은 변수는 인간에게 주어진 문제를 뜻하며 그 문제에 따라 가장 효율적은 자료구조를 선택해야 할 필요성이 생기게 된다. 자료구조가 자료를어떻게 효율적으로 보관할 수 있을지에 대한 고민의 결과물이라면, 알고리즘은 그 자료를 가지고 어떻게 주어진 문제를 효과적으로 풀 수 있을가에 대한 절차적 고민의 결과물이다.
자료구조에 따라 해결 방법이 달라지고 그에따른 문제 해결 속도 또한 달라진다. 주어진 문제에 따라 적합한 자료구조와 알고지름이 달라지기 때문에 세상에서 가장 좋은 자료구조, 알고리즘은 존재하지 않는다. 그때그때 주어진 상황에 맞게 변경되엉야 하는 것이다.
그래서자료구조는 종류가 많으며 그것을 나누는 기준 또한 다양하다. 대표적으로는 형태에 따라 분류를 하는데 크게선형/비선형 자료구조로 분류하며 선형 자료구조에는 스택, 큐, 덱, 리스트 등이 있으며, 비선형 자료구조로는 트리, 그래프, 이진트리, 힙 등이 있다.
스택은 한쪽은 입구, 반대쪽은 출구인 양쪽이 뚫린 파이프라고 생각하면 된다. 그러다보니 먼저 들어온 자료가 먼저 나가는 형태의 자료구조이며 그래서 Firtst In First Out 이라고 FIFO 구조라고 한다. 큐는 스택에서 출구가막힌, 즉 한쪽만 뚫려있는 파이프이다. 그러다보니 먼저 들어온 자료가가장 먼저 나가게 되는 Last In First Out(LIFO) 구조이다. 덱은 큐와 형태는 비슷하나 입구와 출구가 나누어져 있지 않고 양쪽 모두 입구와 출구를 병행하는 형태이다.
리스트는말 그대로 자료들이 서로를 연결하여 하나의 긴 줄처럼 엮여있는 모양이다. 트리는 나무의 가지, 뿌리처럼 생겨서 부모-자식 과 같은 연결구조이며 그래프는 정점과 정점을 잇는 선으로 구성되어 있다. 이진트리는 부모-자식으로구성된 트리에서 자식 노드가 최대두개로만 구성된 트리이며, 힙은 이진 트리에 어떤 특성을 부여하여 데이터를 저장하고 관리하는 형태의 구조이다.
자료구조는 근본적으로 '어떻게 하면 효과적으로 문제를 해결할 수 있을까?' 에서 출발한 개념이기 때문에 빨리 문제를 해결하려는, 즉 얼마나 빨리 연산을 할 수 있는가의 문제가 중요하다. 그래서 자료구조별로 '복잡도'라는 계산을 하게 되고 얼마나 좋은 성능을 나타내는지에 대한 지표로 활용하고 있다. 복잡도는 크게 시간/공간 복잡도로 나누는데 시간 복잡도는말 그대로 '얼마나 빨리 처리하는가?' 에 대한 지표이고 공간 복잡도는 '얼마나 메모리를 많이 잡아먹는가?' 에 대한 지표이다. 앞서 언급했듯 처리 속도가 중요하기 때문에 대부분 시간 복잡도를 중요하게 생각한다. 공간 복잡도는 이 연산이 메모리를 얼마나 썼느냐를 나타내는데 연산을 처리하는 시간 대비, 메모리를 쓰고 지우는 작업은 극히 짧게 걸리기 때문에 일반적으로 시간 복잡도를 많이 쓴다. 게다가 요즘은 반도체 기술의 발달로 메모리의 영역은 충분한게 사실이다. 빅데이터 시대가 도래하긴 하였지만 그래도 아직까지는 처리해야 할 데이터의 양 보다는 메모리의 여유가 더 많은 시대이다. 어떻게 보면 컴퓨터가 수행하는 연산 속도를 비교한다는 자체가 인간에게 있어서 무의미할 정도로 크기 편차가

심하지 않다.(아니 알아볼 수 없다.) 왜냐하면 워낙 빠른 속도로 처리를 하 기때문에 인간이 직접 그 연산 속도를 체험할 수 없기 때문이다. 하지만 객관적으로 알고리즘 간 성능을 비교하기 위해선 비교적 차원의 척정값이 필요하기도 하고연산의 양이 많아지면 때에 따라서사람이 느낄 수 있을 만큼 속도 차이가 나기 때문에 다양한 성능 분석 방법이 존재한다. 그런 시간 복잡도를 나타내는 표기는 대표적으로 'Big O' 표기법이 있다. 함수를 나타내는 표현처럼 'O(n)' 이런식으로 나타내는데 이때 n은 데이터의 양(갯수)을 뜻한다. 즉, 다루어야 하는 데이터를 기준으로 산술적으로 표현한 방법이라고 생각하면 된다. n에는 숫자(상수)가 들어갈 수도 있고, log값, 제곱값 등 각종 산술적인 표현이 들어갈 수 있다. 자료구조와 알고리즘에 따라 대표적으로 O(1), O(log n), O(n^2), O(2^n) 등이 있다. 가장 처음 언급한 상수형은 자료수와 상관없이 일정한 알고리즘이 작동하며 그에 따라 시간복잡도가 동일한 알고르즘이다. 로그형은 자료의 수가늘어남에 따라 복잡도 또한 늘어나는 모습이지만 로그 곡선을 그리면서 완만하게증가하기 때문에 매우 만족스러운 속도를 뽑아낸다. 일반적인 컴퓨터 연산의 경우 데이터의 수가 천문학적으로 많기 때문에 선형적으로 늘어나는 복잡도보다 로그 방식의 복잡도를 띌 때 사람이 크게 알아차리지 못할 만큼의 성능저하를 보이게 된다. n^2의 경우에는 데이터수의 제곱만큼 연산이 되기 때문에 그리 바람직하지 못한 복잡도이며 그보다 더 좋지 않은 복잡도는 2^n 형태, 그리고 가장 복잡한 n!의 형태의 알고리즘이 있다.