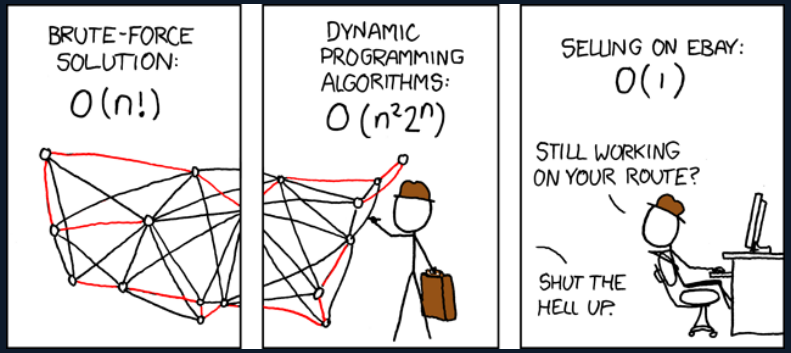
Algorithm Efficiency



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Algorithm Efficiency

- In most cases, there are many ways to accomplish the same task.
  - For example, sorting data, pathfinding, NPC behaviours

- It is often important to be able to estimate how long a particular algorithm will take, given a set of data.
  - Some algorithms will work better than others in a given situation, but perform worse in others in the next.

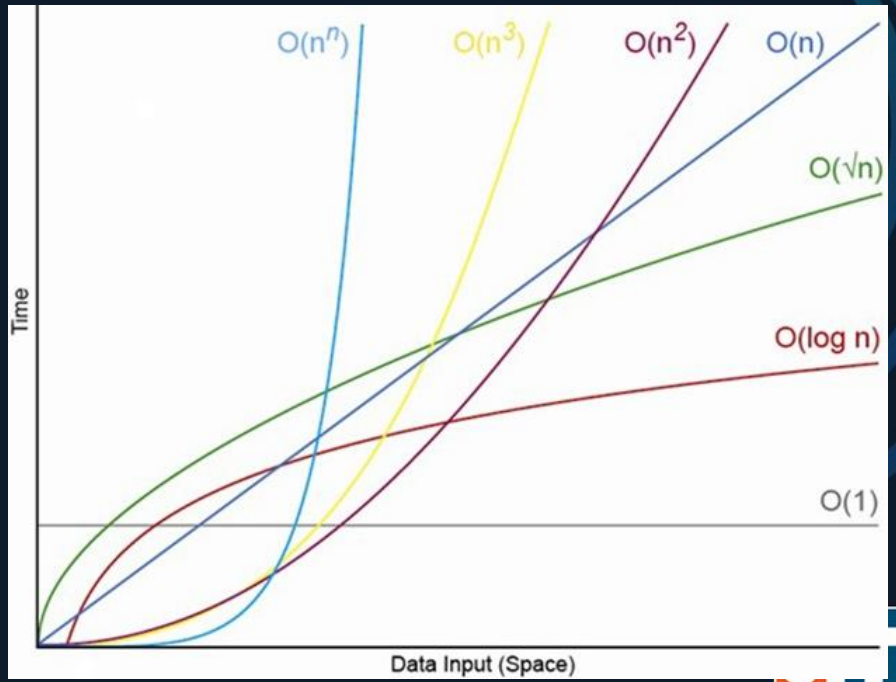
Big O Notation

- “Big O Notation” is a system of explaining the worst case execution times of an Algorithm.

- O means “order”, as in mathematics, and the system describes how the run time increases in relation to the number of items affecting the algorithm.

Big O Notation

$O(1)$	constant
$O(n)$	linear
$O(n^2)$	quadratic
$O(2^n)$	exponential
$O(\log^n)$	logarithmic



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Big O Notation – O(1)

- This operation performs in constant time.
- The operation always takes the same amount of time to run, and is unaffected by the number of items it must operate on.

```
bool IsFirstElementNull(int items[], int nItems)
{
    printf("testing first element of %i items\n", items);
    if (items[0] == NULL)
        return true;
    return false;
}
```

Big O Notation – O(n)

- This algorithm performs in linear time
  - e.g., if *n* doubles, the time doubles

- A search through an unordered array is an example of linear time

```
bool ContainsValue(int items[], int nItems, int searchValue)
{
    for(int i = 0; i < nItems; ++i)
    {
        if (items[i] == searchValue)
            return true;
    }
    return false;
}
```

O(n), cont'd

- In the previous example it is quite possible to return before the entire array is searched, suggesting that the search time is less than O(n)
- Big O notation, however, states the worst case limit

$O(n^2)$

- Quadratic order is far worse than linear,
  it's proportional to the square of the size of the dataset
- Consider the problem of testing if an unsorted array has an entry that appears twice; the following is a simple ('brute-force') implementation

```
bool ContainsDuplicate(int items[], int nItems)
{
    for (int i = 0; i < nItems; i++) {
        for (int j = 0; j < nItems; j++) {
            if (items[i] == items[j])
                return true;
        }
    }
    return false;
}
```

$O(2^n)$

- Exponential growth
- Similar to the previous case, however, the execution time will increase very sharply (much worse than quadratic) as the dataset grows!
- This means the algorithm is very slow

$O(log^n)$

- This is logarithmic time
- Its relative performance actually improves as the n increases
  - in other words, the execution time is linear O(n) for small sets, and tends towards O(1) for large sets
- A binary search is a great example of this. The algorithm continues to halve a dataset till either the search value is found, or the end of the set has been reached

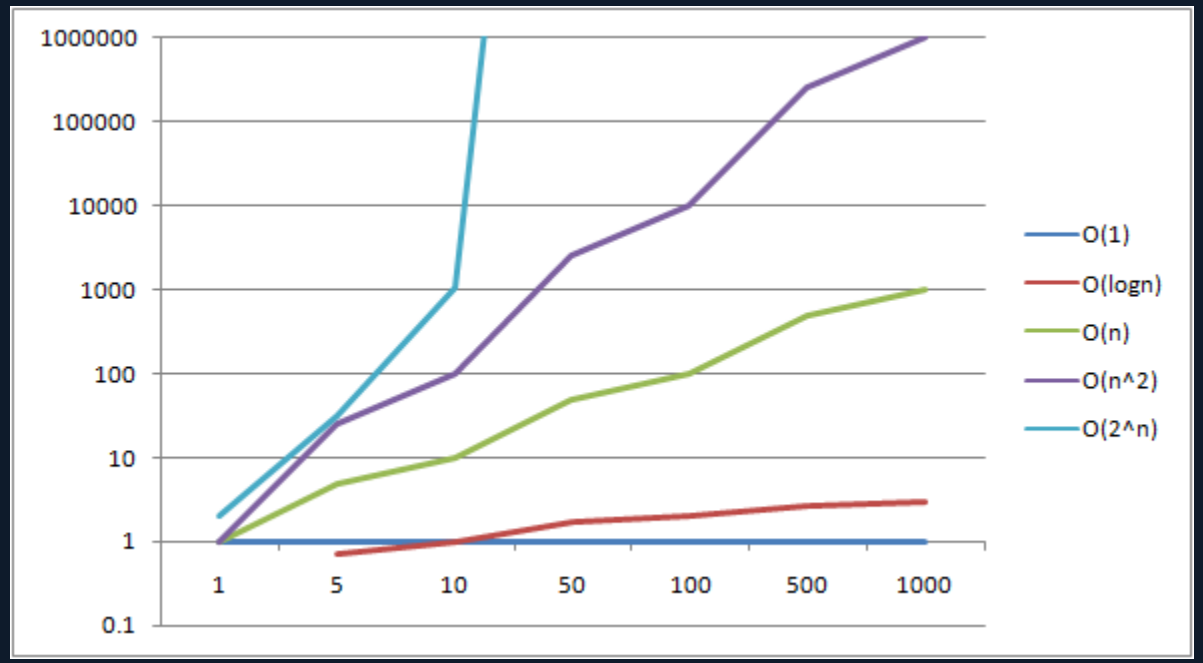
$O(log^n)$

- This does not mean the algorithm takes less time the more items you need to operate on
  - Rather that the time cost of each new item is smaller than the one before it

$O(\log^n)$

```
// search a sorted array (array[i+1] >= array[i])
int BinarySearch(int items[], int nItems, int searchValue)
{
    int start = 0;
    int stop = nItems;
    while (start < searchEnd) {
        int index = (start+stop)/2; // find the average index
        int v = items[index];       // get the value at that index
        if (searchValue > v)
            start = index+1;        // continue search in upper part
        else if (searchValue < v)
            stop = index-1;         // continue search in lower part
        else
            return index;           // searchValue found, return index
     }
    return -1;                      // not found, return fail indicator
}
```

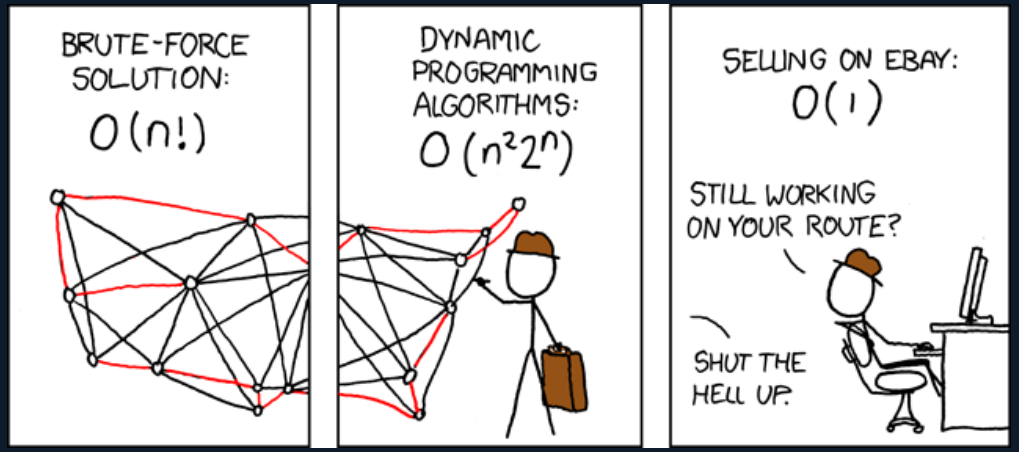
Time vs. Dataset Size



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Make sense now?



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- When using a existing algorithm, research its Big O “score”, and work out whether it is the right one to use
- The running time of a algorithm is based on both the number of items, but also how those items are stored
  - Getting the smallest item in a sorted list is constant – just return the first one!
  - Getting the smallest item in a non sorted list is linear time!
- If writing your own algorithm for something, calculate the Big O score from the pseudocode to ensure it is worth writing in the first place!