Complexity

ZyBook 2.3-2.6

Motivation

```
void dequeue(Queue *psQ, void *pBuf, int size)
{
    // EXIT Node at the end of the list
    ListElementPtr psCurr;
    // ignore error checking
    psCurr = psQ->psList;

   // walk to the end of the list
   while( psCurr && psCurr->next)
   {
      psCurr = psCurr->psNext;
   }

   memcpy(pBuf, psCurr->pData, size);
   free(psCurr->psNext);
   psCurr->psNext = NULL;
}

void dequeue(Queue *psQ, void *pBuf, int size)
{
   // EXIT Node at the head of the list
   ListElementPtr psCurr = psQ->psList;
   memcpy(pBuf, psQ->psList->pData, size);
   psQ->psList = psQ->psList->next;
   free(psCurr);
}
```

```
struct ListElement
{
   void* pData;
   ListElementPtr psNext;
};

struct Queue
{
   ListElementPtr psList;
};
```

Goals

- How much storage is used?
  - space complexity
- How many “steps” are run?
  - runtime complexity
- Based on the size of the input data
  - what does this mean?

Two types of performance

- This algorithm takes N^2 steps to run
  - We care about this!

- This loop takes 10 instructions but I can rewrite it to take 9!
  - We don’t care about this one in this class
  - Later in life, this might be important
  - Many people use this as an excuse to write bad code

Algorithmic Complexity

- How does the runtime increase as the problem size increases?
  - we care about growth, not actual numbers
  - I doubled my input size, how did my runtime grow?
- How do we measure the runtime?
  - language, machine independent!
- How do we measure the problem size?

Runtime Complexity

```
const int N = 500;
int i;
int aCounts[N];

for (i = 0; i < N; ++ i)
{
  aCounts[i] = 0;
}
```

- What value does the running time depend on?
  - changing what number changed the runtime?

Runtime Complexity

```
const int N = 500;
int i, j;
int aCounts[N][N];

for (i = 0; i < N; ++ i)
{
  for (j = 0; i < N; ++j)
  {
    aCounts[i][j] = 0;
  }
}
```

N? Complexity?

```
int isSorted (const int aNums[], int howmany)
{
  int bSorted;
  int i;

  bSorted = true;

  for (i = 0; i < (howmany - 1); ++i)
  {
    if (aNums[i] > aNums[i + 1])
    {
      bSorted = false;
    }
  }

  return bSorted;
}
```

Formal Definition: Big-O

- Computational Complexity
  - number of steps related to some data size, N
  - number of items
  - O(N)
  - $O(N^2)$
- Growth rate!

- Algorithm:

Big-O

- Growth, not exact runtime
- Can’t (always) tell which algorithm is faster
- Concerned with very large inputs
- Asymptotic algorithm analysis

Big-O notation

- Find a function, g(n), that describes the execution time
- O(g(n))

- We only include the highest order terms
  - also ignore constants

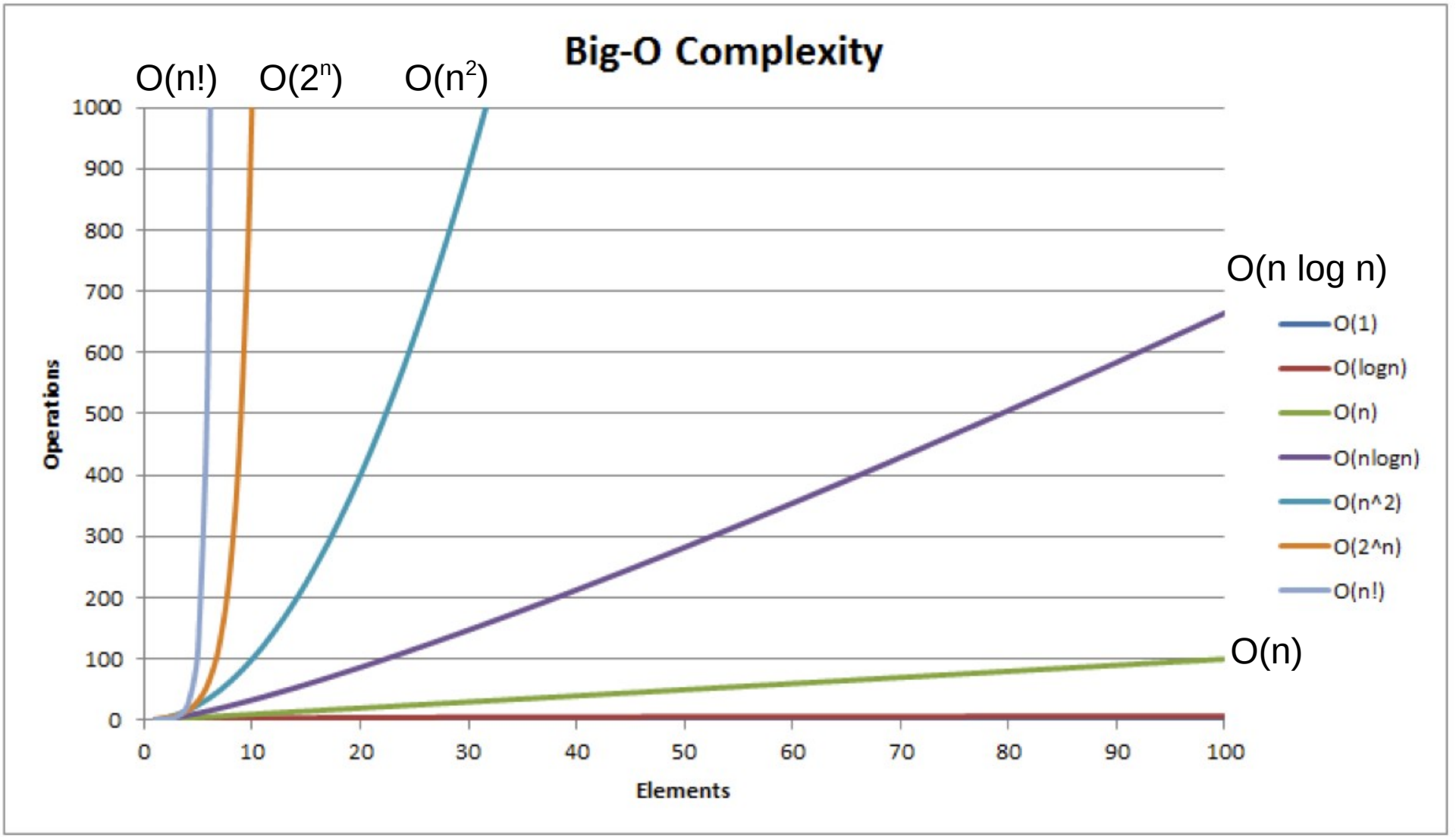
- $x^2 + x + 1$
  - Which term dominates this equation (as x gets big)?

Categories

- $O(1)$ - constant
- $O(\log_2 N)$ - logarithmic
- $O(N)$ - linear
- $O(N\log_2 N)$ - log linear
- $O(N^2)$ - quadratic
- $O(N^3)$ - cubic
- $O(2^N)$ - exponential
- $O(N!)$ - factorial

Growth Rates

N	$\log_2 N$	$N \log_2 N$	N^2	N^3	2^n
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536


Big-O Complexity
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
Operations
Elements
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

Scenarios

- Best Case
- Average Case
- Worst Case

Identify Big-O

	Average case				Worst case		
	Search	insert	Delete		Search	Insert	Delete
Unordered Array							
Ordered Array							
Singly Linked List							

Identify Big-O

Function	Best	Worst	Average
strLength			
strEqual			
strConcat			
strAppend			
strReverse			
strClear			
strCopy			

What is N?

```
typedef struct {
  int length;
  char data[1024];
} String;
```

Formally

Function f(n) is O(g(n)) iff there exist positive constants c and n_0 such that f(n) <= cg(n) for all n, where n >= n_0.

```
for (i = 0; i < howmany; ++i)
{
  for (j = i + 1; j < howmany; ++j)
  {
    if(aNums[i] < aNums[j])
    {
      temp = aNums[i];
      aNums[i] = aNums[j];
      aNums[j] = temp;
    }
  }
}

// how many times is the if() test executed?
```

Compiler

- gcc -O# -o bin/runMe bin/driver.o bin/stack.o
- # is zero to 3
  - zero: default, no speed optimization
  - 1, 2, 3 increasing levels of optimization (speed/size)
    - almost no chance of the debugger working

http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html