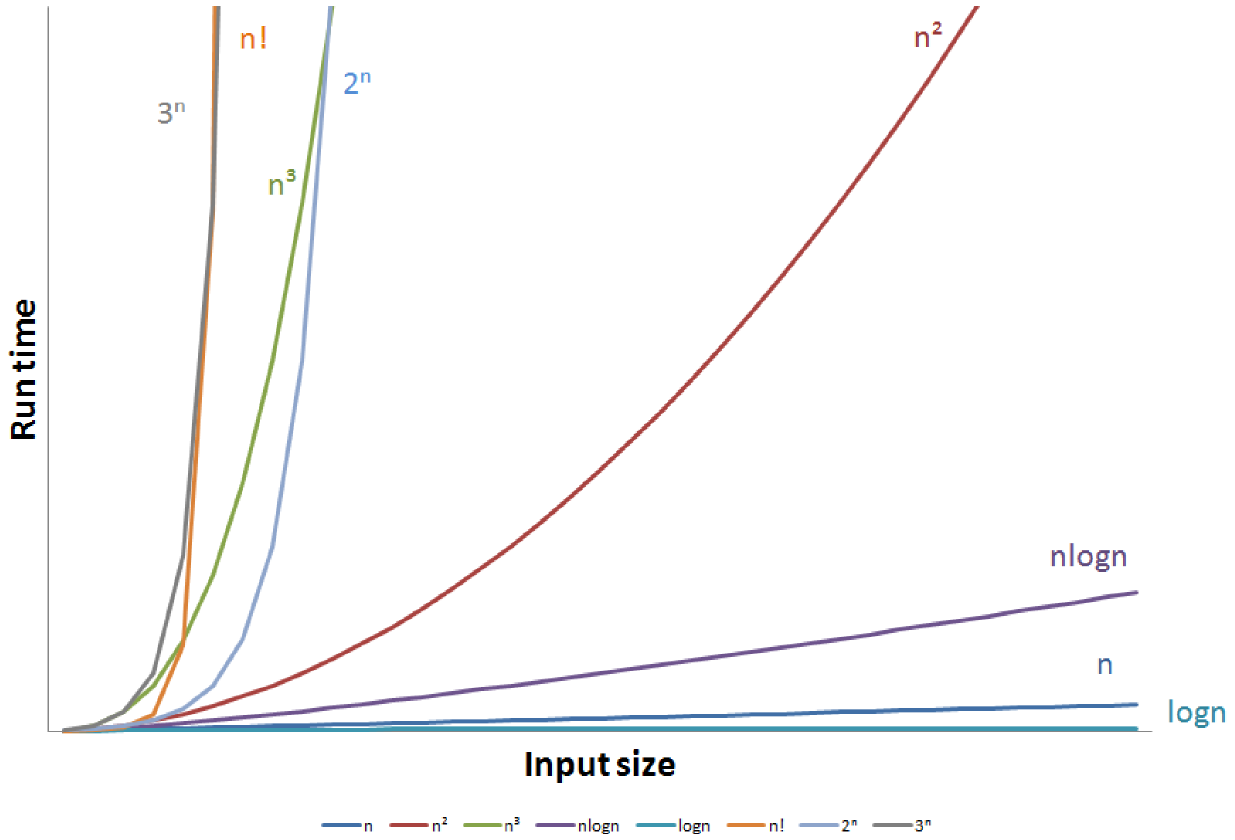
Extra Big-O Examples

Runtime vs. Input Size for common Big-O values



Big-O Examples

#1

Question:

The running time of a function is:

T(n) = n^3 + 20n + 1

What is Big-O?

Answer:

Remove addition of constant 1:	n^3 + 20n
Remove multiplication of constant 20:	n^3
Big-O:	O(n^3)

#2

Question:

What is the Big-O of the isFirstLetterVowel function?

```c
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

int isFirstLetterVowel(const char *str)
{
   const char *VOWELS = "aeiou";
   const int NUM_VOWELS = strlen(VOWELS);
   const char FIRST_LETTER = tolower(str[0]);
   int i;

   for (i = 0; i < NUM_VOWELS; ++i)
      if (FIRST_LETTER == VOWELS[i])
         return true;
   return false;
}

int main()
{
   isFirstLetterVowel("Hello, world!");
   return EXIT_SUCCESS;
}
```

Answer:

```c
int isFirstLetterVowel(const char *str)
{
   const char *VOWELS = "aeiou";
   const int NUM_VOWELS = strlen(VOWELS);        /* O(c1) */
   const char FIRST_LETTER = tolower(str[0]);    /* O(c2) */
   int i;

   for (i = 0; i < NUM_VOWELS; ++i)              /* O(c3) */
      if (FIRST_LETTER == VOWELS[i])             /* O(c4) */
         return true;                            /* O(c5) */
   return false;                                 /* O(c6) */
}
```

Performance:

= O(c1 + c2 + c3 * c4 + (c5 or c6))

= O(1)

Because this function’s runtime is determined only by constant values and does not depend on the size of the input it runs in constant time: O(1)

#3

Question:

What is the Big-O of the containsLetter function?

```
#include <stdlib.h>
#include <string.h>

int containsLetter(const char *str, char letter)
{
   const int STRLEN = strlen(str);
   int i;

   for (i = 0; i < STRLEN; ++i)
      if (str[i] == letter)
         return true;
   return false;
}

int main()
{
   containsLetter("Hello, world!", 'z');
   return EXIT_SUCCESS;
}
```

Answer:

```
int containsLetter(const char *str, char letter)
{
   const int STRLEN = strlen(str);  /* c1 * n */
   int i;

   for (i = 0; i < STRLEN; ++i)     /* c2 * n */
      if (str[i] == letter)         /* c3 */
         return true;               /* c4 */
   return false;                    /* c5 */
}
```

Performance:

= O(c1n + c2n * c3 + (c4 or c5))

= O((c1 + c2)n)

= O(n)

This function's runtime is linearly proportional to the length of str. Therefore the Big-O is the linear runtime O(n).

#4

Question:

What is the Big-O of the containsAny function?

```
#include <stdlib.h>
#include <string.h>

int containsAny(const char *str, const char *possible)
{
   const int STRLEN = strlen(str);
   const int POSSIBLELEN = strlen(possible);
   int i, j;

   for (i = 0; i < STRLEN; ++i)
      for (j = 0; j < POSSIBLELEN; ++j)
         if (str[i] == possible[j])
            return true;
   return false;
}

int main()
{
   containsAny("Hello, world!", "aeiou");
   return EXIT_SUCCESS;
}
```

Answer:

```
int containsAny(const char *str, const char *possible)
{
   const int STRLEN = strlen(str);            /* c1 * n */
   const int POSSIBLELEN = strlen(possible); /* c2 * m */
   int i, j;

   for (i = 0; i < STRLEN; ++i)               /* c3 * n */
      for (j = 0; j < POSSIBLELEN; ++j)       /* c4 * m */
         if (str[i] == possible[j])           /* c5 */
            return true;                      /* c6 */
   return false;                              /* c7 */
}
```

Performance

= O(c1n + c2m + c3n * c4 m * c5 + (c6 or c7))

= O(c3nc4m + c1n + c2m +(c6 or c7))

= O(c3nc4m + c1n + c2m)

= O(c3nc4m)

$= O(nm)$

The function’s runtime is proportional to the product of the sizes of the two input values – the string being searched and the array of characters being searched for.

#5

Question:

What is the Big-O of the containsDuplicates function?

```c
#include <stdlib.h>
#include <string.h>

/* Returns whether the string contains any duplicate letters */
int containsDuplicates(const char *str)
{
   const int STRLEN = strlen(str);
   int i, j;

   for (i = 0; i < STRLEN ; ++i)
      for (j = i + 1; j < STRLEN; ++j)
         if (str[i] == str[j])
            return true;
   return false;
}

int main()
{
   containsDuplicates("Hello, world!");
   return EXIT_SUCCESS;
}
```

Answer:

```c
/* Returns whether the string contains any duplicate letters */
int containsDuplicates(const char *str)
{
   const int STRLEN = strlen(str);        /* c1 * n */
   int i, j;

   for (i = 0; i < STRLEN ; ++i)          /* c2 * n */
      for (j = i + 1; j < STRLEN; ++j)    /* c3 * n / 2 */
         if (str[i] == str[j])            /* c4 */
            return true;                  /* c5 */
   return false;                          /* c6 */
}
```

Performance:

= O(c1n + c2n * c3n/2 * c4 + (c5 or c6))

= O(c2c3c4n^2/2 + c1n + (c5 or c6))

= O(c2c3c4n^2/2 + c1n)

= O(c2c3c4n^2/2)

= O(n^2)

The performance of the function is proportional to the square of the length of the string. This performance is exponential: O(n^2).