CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

Business

- Grading update:
  - PS0 returned
  - Project1 started—grades probably not returned until end of Break
    - (Please let me know if you'd like feedback sooner!)
- I welcome feedback about grader feedback—let's talk!
- Problem Set 1 due Oct. 5
  - See note about *break* statements—please avoid them in CS375
  - Please get started early and ask questions early!
- Another SA may be out soon—I'll email if so
- Revisions on SA1 (where applicable):
  - This is foundational material
  - Please get to them soon, to help with learning for the course

Business, pt. 2

- Office Hours will be cancelled Wednesday, Oct. 5
  - I will not be able to stay late for my Hours on Tuesday, Oct. 4, either
- Class will be held on Wednesday, Oct. 5—please attend

Recall the 2^n entry in our table of common complexity classes

Exponential Time, and The Power Set of a Set S

- When we think of exponential time—or, more generally, something of size / length 2^n—we often think of all subsets of a set of size n

Consider set S of size n. How many subsets of S are there?

As a small example, consider S = {1, 2, 3}; n = 3. What are all the subsets?

{{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}} – there are 8 of them.

The set of all subsets of S is called the *power set* of S.

Exhaustive Search (Brute Force)

- How many subsets are there of a given set *S*?
  - Say, for notation, *S* has *n* elements

Vocab: The set of all subsets of S is called the ***power set*** of S

- The power set of ***S*** has 2^n elements.
- What does this tell us about the asymptotic complexity of an ***exhaustive search*** algorithm over all subsets of a set?
  - Remember, ***exhaustive*** search implies that it looks at all elements (at least in worst case) of a collection
  - Will we describe complexity using Big-O, θ, or Ω?

Exhaustive Search (Brute Force)

- How many subsets are there of a given set *S*?
  - Say, for notation, *S* has *n* elements

Vocab: The set of all subsets of S is called the ***power set*** of S

- The power set of ***S*** has 2^n elements.
- What does this tell us about the asymptotic complexity of an exhaustive search algorithm over all subsets of a set?
- How many orderings (or *permutations*) are there of all elements in a list L = [a_1, …, a_n]?

All Permutations of a List L

Consider list $L = <a_1,a_2,a_3,\ldots,a_n>$ of length n. How many permutations are there of L—i.e., ways to put all the elements into some list, in any order?

As a small example, consider L = <1, 2, 3>; n = 3. What are all the permutations?

<1,2,3>,<1,3,2>,<2,1,3>,<2,3,1>,<3,1,2>,<3,2,1> – there are 6 of them.

How many would there be for list L'=<1,2,3,4>? [Hint: it's not just 4 more]

Recall the n! entry in our table of common complexity classes

Factorial Time, and All Permutations of a List L

- When we think of factorial time—or, more generally, something of size / length *n!*—we often think of all permutations of a list of length *n*

Consider list $L = <a_1,a_2,a_3,\ldots,a_n>$ of length n. How many permutations are there of L—i.e., ways to put all the elements into some list, in any order?

As a small example, consider L = <1, 2, 3>; n = 3. What are all the permutations?

<1,2,3>,<1,3,2>,<2,1,3>,<2,3,1>,<3,1,2>,<3,2,1> – there are 6 of them.

How many would there be for list L'=<1,2,3,4>? [Answer: There are 24, 4 *times* more! Do you see why? Think of how many ways a "4" could be inserted into list <1,2,3> — there are 4 possible places!]

Exhaustive Search (Brute Force)

- How many subsets are there of a given set S?
  - Say, for notation, S has n elements

Vocab: The set of all subsets of S is called the ***power set*** of S

- The power set of S has 2^n elements.
- What does this tell us about the asymptotic complexity of an exhaustive search algorithm over all subsets of a set?

- How many orderings (or *permutations*) are there of all elements in a list $L = [a_1, \ldots, a_n]$?

- A list of length n has $n!$ permutations.
- What does this tell us about the asymptotic complexity of an exhaustive search algorithm over all orderings of elements in a list?
  - Will we describe complexity using Big-O, θ, or Ω?

Exhaustive Search (Brute Force)

- How many subsets are there of a given set S?
  - Say, for notation, S has n elements

Vocab: The set of all subsets of S is called the ***power set*** of S

- The power set of S has 2^n elements.

So, exhaustive search over all subsets is faster than over all permutations. But both are ***really slow!***

- How many orderings (or *permutations*) are there of all elements in a list $L = [a_1, \ldots, a_n]$?

- A list of length n has $n!$ permutations.

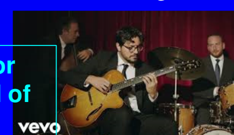
A take-home message: For large n, When things can be ordered [a list], there's a LOT more possibilities than when things can't be ordered [a set].

Now that we've looked at those brute-force measures, let's answer the question here: How many possible itineraries are there?

The CS375 Guitar Genius Tour!

- Guitarists Steve Vai and Pasquale Grasso—both faves of your CS375 Prof.—are *finally going on tour together*!
  - There are n possible venues they could play on their tour
  - They could play any number of them, from 0 to n
  - They have a list ordering of the n venues in mind—the order in which it makes sense to travel to them
    - So, if the ordering is $V_1, V_2, \ldots, V_n$ from first to last, they would never play V_y before V_x if $y > x$. Larger numbers are always later in the ordering.
    - But they could skip, or play, any or all venues

(Disclaimer: They aren't really touring together.)



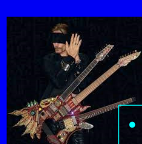
- **We want to figure out the best tour itinerary for them. For now, we'll use a *brute force* method of checking all possible tour itineraries.**
- ***How many possible itineraries are there?***

Now that we've looked at those brute-force measures, let's answer the question here: How many possible itineraries are there?

The CS375 Guitar Genius Tour!

- Guitarists Steve Vai and Pasquale Grasso—both faves of your CS375 Prof.—are *finally going on tour together*!
  - There are n possible venues they could play on their tour
  - They could play any number of them, from 0 to n
  - They have a list ordering of the n venues in mind—the order in which it makes sense to travel to them
    - So, if the ordering is $V_1, V_2, \ldots, V_n$ from first to last, they would never play V_y before V_x if $y > x$. Larger numbers are always later in the ordering.
    - But they could skip, or play, any or all venues

(Disclaimer: They aren't really touring together.)



- **We want to figure out the best tour itinerary for them. For now, we'll use a *brute force* method of checking all possible tour itineraries.**
- ***How many possible itineraries are there? 2^n—one for each set of venues that could be chosen***

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set S, something first needs to generate all subsets of S
  - Let's write an algo that does that!
  - What is its time / space complexity?

```
Generate-All-Subsets(S)
# Input: S, a set of n elements
#   S = s0,s1,s2,…,sn-1
#   [probably implemented as a List,
#   but with no repeated elements, so
#   it can be treated as a set by
#   ignoring the elements' ordering]
# Output: L, a set of all subsets of S
```

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

```
Generate-All-Subsets(S)
# Input: S, a set of n elements
#   S = s0,s1,s2,…,sn-1
#   [probably implemented as a List,
#   but with no repeated elements, so
#   it can be treated as a set by
#   ignoring the elements' ordering]
# Output: L, a set of all subsets of S
```

```
Generate-All-Subsets(S):
   n = len(S)

   # Invariant, outer loop:
   #   Before index i, L contains all subsets formed from elts s0, …, si-1
   #   After index i, L contains all subsets formed from elts s0, …, si
   for i = 0 to n-1
```

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

```
Generate-All-Subsets(S)
# Input: S, a set of n elements
#   S = s0,s1,s2,…,sn-1
#   [probably implemented as a List,
#   but with no repeated elements, so
#   it can be treated as a set by
#   ignoring the elements' ordering]
# Output: L, a set of all subsets of S
```


```
Generate-All-Subsets(S):
   n = len(S)
   L = [ [ ] ] # note relation to invariant below

   # Invariant, outer loop:
   #   Before index i, L contains all subsets formed from elts s0, …, si-1
   #   After index i, L contains all subsets formed from elts s0, …, si
   for i = 0 to n-1
```

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

```
Generate-All-Subsets(S)
# Input: S, a set of n elements
#  S = s0,s1,s2,…,sn-1
#  [probably implemented as a List,
#  but with no repeated elements, so
#  it can be treated as a set by
#  ignoring the elements' ordering]
# Output: L, a set of all subsets of S
```

```
Generate-All-Subsets(S):
  n = len(S)
  L = [ [ ] ] # note relation to invariant below

  # Invariant, outer loop:
  #  Before index i, L contains all subsets formed from elts s0, …, si-1
  #  After index i, L contains all subsets formed from elts s0, …, si
  for i = 0 to n-1
```

A sequence from a bigger number to a smaller number, like $s_0, \ldots, s_{-1}$, is considered empty—containing no elements

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

```
Generate-All-Subsets(S)
# Input: S, a set of n elements
#  S = s0,s1,s2,…,sn-1
#  [probably implemented as a List,
#  but with no repeated elements, so
#  it can be treated as a set by
#  ignoring the elements' ordering]
# Output: L, a set of all subsets of S
```


```
Generate-All-Subsets(S):
  n = len(S)
  L = [ [ ] ] # note relation to invariant below

  # Invariant, outer loop:
  #  Before index i, L contains all subsets formed from elts s0, …, si-1
  #  After index i, L contains all subsets formed from elts s0, …, si
  for i = 0 to n-1
    tempL = [ ] # empty list; will store subsets containing si
    m = len(L)
    # Inner loop: loops over L, create new subsets containing si



    # What's the length of tempL here, when it's fully filled in?
    add all of tempL to L        # What's the length of L after this operation?
```

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

Generate-All-Subsets(S)
\# Input: S, a set of *n* elements
\# $S = s_0, s_1, s_2, \ldots, s_{n-1}$
\# [probably implemented as a List,
\# but with no repeated elements, so
\# it can be treated as a set by
\# ignoring the elements' ordering]
\# Output: L, a set of all subsets of S

Generate-All-Subsets(S):
  n = len(S)
  L = [[ ]] # note relation to invariant below

  \# Invariant, outer loop:
  \# Before index i, L contains all subsets formed from elts $s_0, \ldots, s_{i-1}$
  \# After index i, L contains all subsets formed from elts $s_0, \ldots, s_i$
  for i = 0 to n-1
    tempL = [ ] # empty list; will store subsets containing s_i
    m = len(L)
    \# Inner loop: loops over L, create new subsets containing s_i
    for j = 0 to m-1:
      tempS = deepcopy(L[j]) # why deepcopy?
      add s_i to tempS
      add tempS to list tempL
    \# What's the length of tempL here, when it's fully filled in?
    add all of tempL to L   # What's the length of L after this operation?

Let's go through a small example:

How does this algo work on S=[3,7,5]?

Exponential(-ish): Generate All Subsets

- To search through all subsets of a set *S*, something first needs to generate all subsets of *S*
  - Let's write an algo that does that!
  - What is its time / space complexity?

Generate-All-Subsets(S)
\# Input: S, a set of *n* elements
\# $S = s_0, s_1, s_2, \ldots, s_{n-1}$
\# [probably implemented as a List,
\# but with no repeated elements, so
\# it can be treated as a set by
\# ignoring the elements' ordering]
\# Output: L, a set of all subsets of S

Generate-All-Subsets(S):
  n = len(S)
  L = [[ ]] # note relation to invariant below

  \# Invariant, outer loop:
  \# Before index i, L contains all subsets formed from elts $s_0, \ldots, s_{i-1}$
  \# After index i, L contains all subsets formed from elts $s_0, \ldots, s_i$
  for i = 0 to n-1
    tempL = [ ] # empty list; will store subsets containing s_i
    m = len(L)
    \# Inner loop: loops over L, create new subsets containing s_i
    for j = 0 to m-1:
      tempS = deepcopy(L[j]) # why deepcopy?
      add s_i to tempS
      add tempS to list tempL
    \# What's the length of tempL here, when it's fully filled in?
    add all of tempL to L   # What's the length of L after this operation?

What *is* the time / space com-plexity of this algo?

Factorial(-ish): All Permutations

- Write an algo to generate all permutations of an input list L
  - What are its input / output specifications?
  - How does your algo solve the problem?

What loop invariant would be helpful, to clarify / explain your algorithm design?

  - What is its time / space complexity?

This will be assigned to you as a Smaller Assignment, due after Break