Computer Science Department,
College of Computer and Information Sciences,
King Saud University.

CSC 512: Design and Analysis of Algorithms[1]

Dr. Waleed Alsalih

3.1 Brute force algorithms [Sections 3.1 and 3.2]

Our first algorithm design technique is **brute force**. Brute force is a straightforward approach to solve a problem in a very easy, yet (usually) inefficient way. It is a Just-Do-It approach. To solve a problem using the brute force technique, you just need to use facts derived from the problem statement without making an effort to make the algorithm smart.
Example:
Consider the problem of telling whether a positive integer is a prime number or not.

Algorithm 1: Telling whether a positive integer is prime or not.

Algorithm Prime(n)
if $n \leq 1$ **then**
    return False;
end
i:=$n-1$;
while $i >$ *1 AND n mod i* > 0 **do**
    i:=$i-1$;
end
if $i > 1$ **then**
    return False;
else
    return True;
end

[1] This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Example: Selection sort

Algorithm 2: Selection Sort.

Algorithm SelectionSort($A[0..n-1]$)
for *i:=0..n-2* **do**
  min:=*i*;
  for *j:=i+1..n-1* **do**
    if $A[j] < A[min]$ **then**
      min:=*j*;
    end
  end
  swap $A[i]$ and $A[min]$;
end

The major operation count is $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2}(n-1-i) = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Example: Bubble sort

Algorithm 3: Bubble Sort.

Algorithm BubleSort($A[0..n-1]$)
for *i:=0..n-2* **do**
  for *j:=0..n-2-i* **do**
    if $A[j+1] < A[j]$ **then**
      swap $A[j]$ and $A[j+1]$;
    end
  end
end

The major operation count is $\sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2}[(n-2-i)+1] = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Example:
Consider the string match problem in which you are given a string called text and anothe string called the pattern, and you are asked to tell whether a substring of the text matches the pattern. The algorithm is supposed to return the index of the leftmost character that starts a matching substring if such a matching substring exists and -1 otherwise. The major

Algorithm 4: Brute force string matching.

```
Algorithm StringMatching(T[0..n − 1], P[0..m − 1])
for i:=0..n-m do
    j:=0;
    while j < m AND P[j] == T[i + j] do
        j:=j + 1;
    end
    if j == m then
        return i;
    end
end
return-1;
```

operation count is $(n - m + 1)m \in \Theta(nm)$.

3.2 Exhaustive search [Sections 3.4]

Exhaustive search is a brute force algorithm design technique in which all elements in the search space are generated, and then all elements are checked to find a particular element that represents the answer. Exhaustive search is composed of the following steps:

1. Generate all elements in the search space.
2. Exclude any invalid element.
3. Find a desired element representing the right (best) answer.

Question: Let's say that we have a set $A = \{E_0, E_1, \ldots, E_{n-1}\}$. How can you find all subsets of A?

Answer: We can have 2^n subsets, each of which can be represented by an integer between 0 and $2^n - 1$. Let $B(s)$ be the binary representation of an integer that corresponds to a subset s. $B(s)$ has n binary bits numbered from 0 to $n-1$, and the i^{th} binary bit corresponds to E_i.
The i^{th} bit in $B(s)$ is set to 1 if and only if $E_i \in s$, $0 \leq i \leq n-1$.
Example:
$A = \{Ali, Mohammed, Ahmed\}$
$s = \{Ali, Ahmed\}$
$B(s) = 101$

Question: Given an integer t, how to tell whether the i^{th} bit of the binary representation of t is 1 or 0?
Answer: Left as an exercise!

Example:
Knapsack problem: You have n items and each item i has a weight $W[i]$ and a value $V[i]$. You have a knapsack of capacity C. You want to find the most valuable subset of items that fit into the knapsack.

Algorithm 5: Finding the b^{th} bit of the binary representation of an integer t.

Algorithm BinaryBit(int t, int b)
This algorithm returns the value of the b^{th} bit in the binary representation of an integer t.

This algorithm runs in $O(n \cdot 2^n)$ time.

Algorithm 6: Brute force knapsack algorithm.

```
Algorithm Knapsack(W[0..n-1], V[0..n-1], C)
MaxValue = 0;
MaxSubset = 0;
for i:=1..2^n - 1 do
    Value:=0;
    Weight := 0;
    for j:=0..n-1 do
        if BinaryBit(i,j)==1 then
            Value := Value + V[j];
            Weight := Weight + W[j];
        end
    end
    if Weight <= C AND Value > MaxValue then
        MaxValue = Value;
        MaxSubset = i;
    end
end
return MaxSubset;
```