Math 251, Mon 5-Oct-2020 -- Mon 5-Oct-2020
Discrete Mathematics
Fall 2020

Monday, October 05th 2020

Due:: PS07

Other calendar items

Monday, October 5th 2020

Wk 6, Mo
Topic:: Algorithms
Read:: Rosen 3.1

Algorithms

Properties

- specified set of inputs (domain)
- every input produces output from codomain
- definiteness: clear process to follow
- correctness: accurately finds correct output for each input
- finiteness: desired output is produced after finite number of steps
- effectiveness: possible to do each step in finite amt of time
- generality: applicable to all problems of desired form

Note: Not all problems are solvable in the sense of having an algorithm as described above.

Example: Halting problem. At least one problem is unsolvable.

What is sought in the halting problem: An algorithm that can decide, given any computer program and set of inputs, whether the program halts in finitely many steps. Suppose such a procedure exists, and write

$$H\colon \{\text{programs}\} \times \{\text{inputs}\} \to \{\text{"halts"}, \text{"DNH"}\}.$$

Note: $H(P, P)$ is defined and will have either the value "halts" or "DNH". Define a procedure K which takes programs P as inputs, and

loops forever ("DNH") if $H(P, P)$ halts.
"halts" if $H(P, P)$ DNH.

Notice that $H(K, K)$ can produce either of two values, but both are contradict the behavior of $K(K)$.

Specific algorithms

```
- find smallest element in a list
- search for a key
    in a list (general)
    in a sorted list
- sort a list
    bubble sort
    insertion sort
- optimization via greedy algorithm
    task: make change for n using denominations of size c_1 > c_2 > ... > c_r
    goal: use the fewest number of coins possible
    describe a greedy algorithm approach
    Note: algorithm doesn't always find an optimal solution
      counterexample: n=30,  c1=25, c2=10, c3=1
    However, if nickles are possible, it can be shown the soln is optimal
```

```
1. Search (unordered list) for a key

inputs:  an array a with n elements  a[0], a[1], a[2], ..., a[n]
         key k to find in the array

  for i = 0 to n
    compare k with a[i]
    if match exit with i as the return value
    otherwise loop
  return (-1)

2. Search (ordered list) for a key: binary search

inputs:  an array a with n elements  a[0] <= a[1] <= a[2] <= ... <= a[n]
         key k to find in the array

How it works when searching a list:    2, 5, 7, 11, 18, 36, 51, 55  for key = 17
   splits full list into 2 parts:  {2, 5, 7, 11}  {18, 36, 51, 55}
   decides which "half" list might contain the key (17)
   recursively deals with that half-list

   input: sorted list (length n), key
   if n==1
     if key == a[0]
       return (there is a match)
     else
       return (no match in the list)
   else
     let m = floor(n/2)
     if a[m] >= key
       recursively call our algorithm with the lower half-list
     else
       recursively call our algorithm with the upper half-list

issue of recovery---finding index to return---see Rosen

3. Sorting: bubble sort

  input: array a[0], a[1], a[2], ..., a[n]  to sort from lowest to highest

How it should process the list of numbers:  22, 8, 3, 15, 11, 17, 4

  start list:  22, 8, 3, 15, 11, 17, 4
  compare 22 with 8: 22 > 8  ==>  8, 22, 3, 15, 11, 17, 4
  compare 22 with 3: 22 > 3  ==>  8, 3, 22, 15, 11, 17, 4
  compare 22 with 15: 22 > 15  ==>  8, 3, 15, 22, 11, 17, 4
  compare 22 with 11: 22 > 11  ==>  8, 3, 15, 11, 22, 17, 4
  compare 22 with 17: 22 > 17  ==>  8, 3, 15, 11, 17, 22, 4
  compare 22 with 4: 22 > 4  ==>  8, 3, 15, 11, 17, 4, 22
 one pass, largest value is at the end of the list
 6 total comparisons in that pass

  next: repeat this process on paired down list 8, 3, 15, 11, 17, 4
   2nd pass ==>  17 to end of paired-down list (comparison/trade with 22 does no
t occur)
   5 total comparisons

  next: repeat this process on 2nd paired-down 8, 3, 15, 11, 4
   3rd pass ==>  15 to end of this paired list, with 17 and 22 still right of it
   4 total comparisons

  etc., until paired-down list is of length 1

6 + 5 + 4 + 3 + 2 + 1 = 21  comparisons for entire process
```

```
If starting list had n elements:

 (n-1) + (n-2) + (n-3) + ... + 1    (sum of terms in an arithmetic sequence)
```