COMP20005 Workshop Week 12

Divide & Conquer: Recursion is fun

What's Next?

Other Topics/Exercises

Sample Quiz 3.

Lab: finish implementation of all of the discussed exercises, especially Ex permutations, subset sum, 9.3, 9.7
Additional: Implement 9.11 if you love Physics

DIVIDE AND CONQUER!

Review: Recursion for a task of size n

To apply recursion:

- express the task of size n through *the same tasks*, but of smaller sizes
- find the base cases where solutions are trivial or easy

To write a recursive function, normally:

- deal with the base cases first
- then, deal with the general cases.

Programming is fun
When recursion is done

Examples:

- `n!`
- sum of $(i/i+1)^i$
- Hanoi Tower
- Binary search in sorted array
- Permutation: prints all permutations of numbers from 1 to n
- subset sum: Given a set of numbers (as an array), print, if any, a subset that adds up to k
- in assignment 2: find number of areas that have luminance < 1.0

- Should we try one? Which one?

recursion examples

What's next

In a short term:

- Next Quiz this Friday, marks= 15
- Exam: marks= ?? date=

In a longer term:

If Programing Is Fun:

- consider comp20007 in semester 2, or
- (harder, and eligible?) comp20003 semester 1 next year

And in instant term (of the current workshop):

- implement the quiz's programming questions
- implement 1 or 2 recursive exercises
- implement 9.3
- implement 9.7 (fast version)
- implement 9.11 (if you really love Physics)

Ex 9.3: Simulation?

Write a program that deals four random five-card poker hands from a standard 52-card deck. You need to implement a suitable "shuffling" mechanism, and ensure that the same card does not get dealt twice.

Then modify your program to allow you to estimate the probability that a player in a four-person poker game obtains a simple pair (two cards with the same face value in different suits) in their initial hand. Compute your estimate using 40,000 hands dealt from 10,000 shuffled decks.

How about three of a kind (three cards of the same face value)?

And a full house (three of a kind plus a pair with the other two cards)?

For example:

```
./program
player 1: 3-S, Ac-C, Qu-D, 4-H, Qu-H
player 2: 10-C, 2-H, 5-H, 10-H, Ki-H
player 3: 2-C, 6-D, 10-D, Ki-D, 9-H
player 4: 8-S, 9-S, 10-S, Qu-S, 4-D
Over  40000 hands of cards:
 19680 (49.20%) have a pair (or better)
   900 ( 2.25%) have three of a kind (or better)
    48 ( 0.12%) have a full house (or better)
```

e9.3: let's simulate a game

```
#define FACES      13
#define SUITS       4
#define CARDS (FACES*SUITS)  /* number of cards */

#define PLAYERS     4
#define CARDSINHAND 5

const char *faces[] = {“Ac", “2", “3", “4", “5”, “6”,
                  “7", “8", “9", "10", ”Je", ”Qu", ”Ki"};
const char suits[] = {'S', 'C', 'D', 'H'};

typedef struct {
    int face, suit;       // index to the above arrays
} card_t;

card_t players[PLAYERS][CARDSINHAND];
card_t deck[CARDS];
```

e9.7: The maximum subarray problem

Problem: Given `int a[n]`, *find* l *and* r *so that the sum of elements from* l *to* r *is the maximal possible.*

Note 1: Array `a[]` *can also be of type* `float` *or* `double`.
Note 2: since the sum of zero-length sequence is 0, *if the array contains only negative elements, the answer is a zero-length sequence (for example,* l=0 *and* r= -1*).*

Implement a fast solution.

e9.7: solution2 – mimic a smart manual job

```
2  1  3  -4  -5  2  6  1  -8  9  2  3  1  5  -7  1  -2
-----       ------------------    --
```

```
start with maxsum= 0 (solution= empty subarray)

Checking: which pair (i,j) gives better maxsum?
For the above array:
     i=0 j=0, sum= 2 (new maxsum)
         j=1, sum= 3 (new maxsum)
         j=2, sum= 6 (new maxsum)
         j=3, sum= 2 NO new maxsum
              but should continue because …
         j=4, sum= -3 should stop increasing j...
     should we start again from i=1?
     Nope! what should be new starting value for i?

Should we start from i=0 if A[0]= -1 ?
Build a fast solution by developing the above points.
```

?