CSE250, Spring 2013 Prelim I Name:________________________
Mar. 4, 2013 St.ID#:________________________

Closed book, closed-notes-except-for-1-sheet, closed neighbors, 48 minutes. This question paper has FOUR pages. Do ALL THREE questions **on this exam paper**. Please *show all your work*; this may help for partial credit. The exam totals 67 pts., subdivided as shown.

(1) (13 pts. total)

Order the following functions according to increasing asymptotic running time, with the fastest time first.

(a) $f_a(n) = 4n^3(\log n)$

(b) $f_b(n) = 16n^2(\log n)^2$

(c) $f_c(n) = n^4$

(d) $f_d(n) = 64n(\log n)^3$

(e) $f_e(n) = 256(\log n)^4$.

Finally answer this: **Note that the functions have a common ratio of $4(\log n)/n$, where we may assume the logarithms are to base 2. What is the tradeoff-point value of n where they are all equal? (This can be solved without a calculator—think of some powers of 2.)**

(2) (30 pts. total)

Every letter-indicated line in the following code has an error on it. Identify the error, and show as best you can how to fix it.

```
#include <string>
#include <vector>
#include <iostream>
using namespace std;

template <typename T>   //REQ: T has default construction, so T() is legal
class Stacky {

   vector<T>* elements; //pointer to vector of T
   size_t topSpace;     //CLASS INV: denotes first free space

 public:
   explicit Stacky<T>(size_t guaranteedCapacity) :
      elements(new vector<T>(guaranteedCapacity)), topSpace(0) { }

   size_t size() const { return topSpace; }


   //In the above lines there are *no errors*; lines for the question begin here.

   bool empty() { return topSpace == 0; }                                   //(a)

   bool full() const { return topSpace == elements.size(); }                //(b)

   void push(const T& newItem) {
      if (!(full())) {
         elements->at(topSpace++) = &newItem;                               //(c)
      }
   }

   T pop() const {                                                          //(d)
      if (!(empty())) {
         return elements[--topSpace];                                       //(e)
      }
   }                                                                        //(f)
};


int main() {
   vector<int> myvec = { 2, 6, 13, 4, -7 };  //is legal (only!) in C++11
   Stacky<int> myStack(20);
   const Stacky<int>* myStackpcd = &myStack; //these three lines have no errors
                                             //but now every line is wrong
   myStack = 30;                                                            //(g)
   myStack.push(myvec);                                                     //(h)
   myStack->push(88);                                                       //(i)
   myStackpcd->push(88);                                                    //(j)
   string mysize = myStackpcd->size();                                      //(k)
   int v2 = myvec->at(2);                                                   //(l)
   System.out.println("We're not in Kansas anymore!" + v2 + myStack.pop());//(m)
}
```

(For a 1-pt. extra bonus, even the closing curly brace of `main` is wrong—premature, that is. Why? Also line (m) has possible extra credit for multiple potential errors. Please do all your work on these pages.)

(3) (24 pts. total)

Complete the following code for a function `deal2` which simulates the act of dealing two hands of cards face down, then putting the first hand atop the other. This is the opposite of shuffling the cards, and also reverses them so that the second card dealt winds up on the bottom. The effect is best conveyed by examples. Suppose the deck has seven cards in order: `2 5 6 13 4 7 10`. Thinking of left as "up" and right as "down," we deal the two hands as

```
10 4 6 2
7 13 5
```

Then we put them back together as

```
10 4 6 2 7 13 5
```

For another example, `2 3 4 5 6 7 8 9` becomes `8 6 4 2 9 7 5 3`. If there are 1 or 0 cards there is no change.

The function uses the templated `StackPair` class from Assignments 4 and 6, this time with integers. Call its methods `emptyA`, `emptyB`, `pushA`, `pushB`, `popA`, `popB`, and assume the number of cards won't exceed any capacity limit. Points on this question include handling odd-even and small numbers of cards correctly—logic comments may help for partial credit if they would help debug an error. Here is the code to complete:

```
vector<int> deal2(const vector<int>& deck) {
   vector<int> result;  //initially empty, later returned by value
   StackPair* sp = new StackPair();


     //your code goes here


   while (!(sp->emptyA()) { result.push_back(sp->popA()); }
   while (!(sp->emptyB()) { result.push_back(sp->popB()); }
   return result;
}
```