EECS 477. HOMEWORK 5 SOLUTIONS.

1. Selection with pseudomedian (20pts)

In this problem we ask you to perform analysis of the selection algorithm. We know that when the chunk size 5 is used, the runtime of the algorithm in the worst case is $\Theta(n)$ where n is the size of the input array. Replicate the analysis of section 7.5 as much as you can when the chunk size is equal to 3. Repeat with the chunk size equal to 7. What is the worst runtime asymptotics in these two cases?

You may use the following recursive version of the algorithm in your analysis.

```
float rec_selection(A, i, j, s) {
    unsigned p = pseudo_median(A, i, j);
    pair<unsigned, unsigned> kl = pivot(A, i, j);
    unsigned k = kl.first, l = kl.second;
    if(s<=k)
        return rec_selection(A, i, k, s);
    else if(s>=l)
        return rec_selection(A, k, j, s);
    else
        return p;
}

float pseudo_median(A, i, j) {
    unsigned n = j-i+1; // num of elements in the range
    if(n<=CHUNK_SIZE)
        return adhoc_median(A, i, j);
    unsigned nz = n/CHUNK_SIZE+1;
    vector<float> Z(nz);
    for(int k=0; k<nz; ++k)
        Z[k] = adhoc_median(A, i + CHUNK_SIZE*k,
                            i+CHUNK_SIZE*k+CHUNK_SIZE-1);
    return rec_selection(Z, nz/2);
}
```

Solution. From the above code we can figure out the recurrence relation for the runtime of `rec_selection` procedure on input of length n when the chunk size is equal to $H = 2h - 1$ where h is a positive integer.

$$T(n) \le Cn + T(\lfloor n/H \rfloor + 1) + max\left\{T(m) | m \le \frac{(3h-2)n + h(2h-2)}{2(2h-1)}\right\}$$

Consider $h = 2$ (that is chunk size 3).

$$T(n) \leq Cn + T(\lfloor n/3 \rfloor + 1) + \max\left\{T(m)|m \leq \frac{2n+2}{3}\right\} \tag{1}$$

It is no longer true that $T(n) \leq Kn$ however we can prove that $T(n) \leq Kn \log n$. Without going into the details, the induction step works by substituting the $T(m) \leq Km \log m$ into the right hand side of the recurrence inequality (1):

$$T(n) \leq Kn \log n + Cn + \frac{4}{3}K \log n + KA - K[\log 3 - \frac{2}{3}\log 2]n$$

In order to obtain the above we did need to use the following important inequality

$$\log(n + \alpha) \leq \log n + \frac{\alpha}{n \ln 2}.$$

One can that this is true since $\log(n + \alpha) - \log n = \log(1 + \alpha/n)$, and then the whole logarithm curve lies beneath a straight line with an appropriate slope.

It is clear that by making K big we can make the last negative term dominate all the other ones except for $Kn \log n$ so that we get

$$T(n) \leq Kn \log n,$$

for some large fixed K and for all sufficiently large n.

Now, for $h = 4$ (that is chunk size 7), the arguments proving $T(n) \leq Kn$ go through, so that we get

$$T(n) \leq Cn + T((n + 7)/7) + \max\{T(m)|m \leq (5n + 12)/7\}.$$

Then in the inductive step we get $T(n) \leq 19/7 + (C + 6K/7)n$ so that there is a fixed large K such that $T(n) \leq Kn$.

In order to get full credit you would need to show that the linear runtime argument works for chunksize 7 and does not work for chunksize 3. The $n \log n$ result for chunksize 3 is extra.

2. Knapsack (20pts)

Solve problem 9.54 from the book.
Solution. Denoting by $V[i, w]$ the optimal solution with only the first i types of objects and bound of w, we can distinguish two kinds of optimal collections: ones containing objects of type i and ones that do not. This brings us the following optimality principle:

$$V[i, w] = \max\{V[i - 1, w], V[i, w - w_i] + v_i\}.$$

Then the table is built like this:

bound

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	1	1	2	2	3	3	4	4	5
0	0	1	**3**	3	4	6	6	7	9	9
0	*0*	1	**3**	5	*5*	6	8	10	*10*	11
0	0	1	**3**	5	5	6	9	10	*10*	**12**

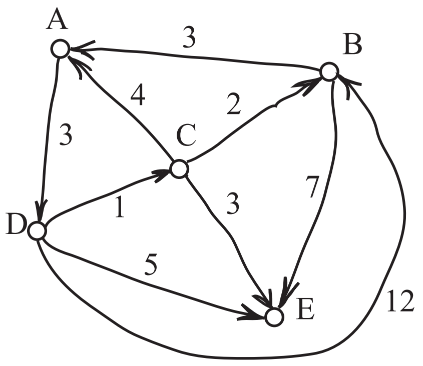
weight	value
2	**1**
3	**3**
4	**5**
7	**9**

The branch and bound trace would look something like that below:

```
Items are chosen in non-increasing value per unit weight order which
coincidentally is also non-increasing weight order.
Notation:([chosen item weights], current_value, <=current_bound)
start:([],0, <=90/7)
choose 7
([7],9, <=9 + 3*9/7=90/7)
best value 9
choose 3
([7+3],9+3=12, <=12)
best value 12
backtrack
choose 2
([7+2],9+1=10, <=10+1/2 = 10.5) bound worse than known best
cull the branch
backtrack
backtrack
choose 4
([4],5,<=5+6*5/4=12.5
choose 4
([4+4], 10, <=12.5
choose 2
[4,4,2], 5+5+1=11,<=11
best is still the same as before
backtrack
backtrack
choose 3
[4,3],8, <=8+3*1=11 worse than known
cull
backtrack
choose 2
[4,2], 6, <=8 worse than known
backtrack
backtrack
choose 3
[3], 3, <= 4+7*1 = 10 worse than known
cull
backtrack
```

```
choose 2
[2], 1, <= 1+8*1/2=5 worse than known
cull
backtrack
END: optimal is [7+3] with value 12.
```

3. Floyd's algorithm (20pts)



For the directed graph above compute the matrix of shortest path distances using Floyd's method; show all the intermediate matrices.

Matrices shown below:

$$\begin{bmatrix}0&\infty&\infty&3&\infty\\3&0&\infty&\infty&7\\4&2&0&\infty&3\\\infty&12&1&0&5\\\infty&\infty&\infty&\infty&0\end{bmatrix}\begin{bmatrix}0&\infty&\infty&3&\infty\\3&0&\infty&6&7\\4&2&0&7&3\\\infty&12&1&0&5\\\infty&\infty&\infty&\infty&0\end{bmatrix}\begin{bmatrix}0&\infty&\infty&3&\infty\\3&0&\infty&6&7\\4&2&0&7&3\\15&12&1&0&5\\\infty&\infty&\infty&\infty&0\end{bmatrix}$$

$$\begin{bmatrix}0&\infty&\infty&3&\infty\\3&0&\infty&6&7\\4&2&0&7&3\\5&3&1&0&4\\\infty&\infty&\infty&\infty&0\end{bmatrix}\begin{bmatrix}0&6&4&3&7\\3&0&7&6&7\\4&2&0&7&3\\5&3&1&0&4\\\infty&\infty&\infty&\infty&0\end{bmatrix}\begin{bmatrix}0&6&4&3&7\\3&0&7&6&7\\4&2&0&7&3\\5&3&1&0&4\\\infty&\infty&\infty&\infty&0\end{bmatrix}$$

4. Coins (20pts)

(A) Apply the dynamic programming algorithm to the problem of paying \$17 within the system of coinage with coins in denominations \$1, \$4, \$7 available. Fill the table, and get the answer.

Solution. The optimality principle is

$$V[i, s] = \min\{V[i-1, s], V[i, s-d_i] + 1\}.$$

The table is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
\$1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
\$4	0	1	2	3	1	2	3	4	2	3	4	5	3	4	5	6	4	5
\$7	0	1	2	3	1	2	3	1	2	3	4	2	3	4	2	3	4	5

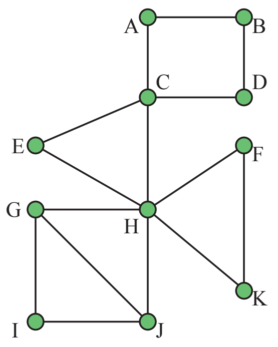
The possible optimal solutions then could be $1+4+4+4+4$, $1+1+1+7+7$, $1+1+4+4+7$.

(B) Write pseudocode for the recursive version of dynamic programming coin payment algorithm that uses a memory function.

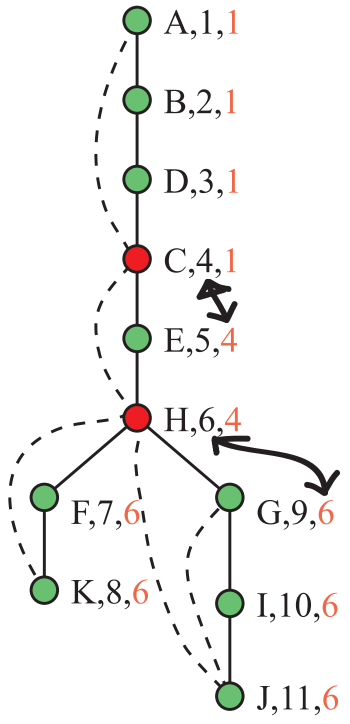
```
//initialize mtab to -1 at first

int coin(int i, int s) {
    if( s<0 OR i<1 )
        return +INFINITY;
    if(s==0)
        return 0;
    if(mtab[i,s]<0) {
        mtab[i,s] = max( coin(i-1,s), coin(i,s-d[i])+1 );
    }
    return mtab[i,s];
}
```

5. DFS(20pts)



For the graph above perform depth-first search starting with vertex A and draw the corresponding spanning tree (together with the remaining graph edges as dashed lines). Index the graph nodes as in preorder traversal. Find the highest index for every vertex as defined in the book (page 297). Find the articulation points using criteria from the book.



6. EXTRA 20 PTS

Do problem 7.38 from the book.
The code below does it.

```
void tour_even(int i, int j, vector< vector< int > >& days) {
  int n = j-i;

  if(n<=1)
    return;

  assert(n%2==0);

  int half = n/2;

  vector< vector< int > > days_left, days_right;
  if(half%2==0) {
    tour_even(i, i+half, days_left);
    tour_even(i+half, j, days_right);

    for(int k=0; k<days_left.size(); ++k) {
      days.push_back(vector<int>());
      days.back().insert(days.back().end(),
                         days_left[k].begin(), days_left[k].end());
      days.back().insert(days.back().end(),
                         days_right[k].begin(), days_right[k].end());
    }

    for(int k=0; k<half; ++k) {
      days.push_back(vector<int>(n, -1));
      for(int s=0; s<half; ++s) {
        int ss = (s+half-k)%half;
        days.back()[ss] = i+half+s;
        days.back()[half+s] = i+ss;
      }
    }
  } else {
    tour_odd(i, i+half, days_left);
    tour_odd(i+half, j, days_right);

    for(int k=days_left.size()-1; k>=0; --k) {
      days.push_back(vector<int>());
      int kk = days_left.size()-1-k;
      days_left[k][kk] = i+half+kk;
      days_right[k][kk] = i+kk;
      days.back().insert(days.back().end(),
                        days_left[k].begin(), days_left[k].end()-1);
      days.back().insert(days.back().end(),
                        days_right[k].begin(), days_right[k].end()-1);
    }


    for(int k=(half==1 ? 0 : 1); k<half; ++k) {
      days.push_back(vector<int>(n, -1));
      for(int s=0; s<half; ++s) {
        int ss = (s+half-k)%half;
        days.back()[ss] = i+half+s;
        days.back()[half+s] = i+ss;
      }
    }
  }
}

void tour_odd(int i, int j, vector< vector< int > >& days) {
  if(j-i<=1)
    return;
  tour_even(i, j+1, days);
  for(int d=0; d<days.size(); ++d)
    days[d][days.size()-1-d] = -1;
}

int main(int argc, char* argv[]) {
  if(argc<2)
    return -1;
  int n = atoi(argv[1]);
  cerr << "tournament with " << n << " players" << endl;
  vector< vector< int > > days;
  if(n%2==0)
    tour_even(0, n, days);
  else
    tour_odd(0, n, days);

  for(int d=0; d<days.size(); ++d) {
    for(int i=0; i<days[d].size(); ++i)
      cerr << days[d][i] << " ";
    cerr << endl;
  }
  return 0;
}
```