Tamar Hariy

# Data Structures #3

③ ⓐ singly linked list with a pointer to the first frenzied task, a pointer to the last frenzied task and a pointer to the last regular task.

ⓑ
```
void insert (Task t)
{
    node *n = new node;
    n->data = t;
    if (t.code = 1)
    {
        n->next = manager.head;

        manager.head = manager.firstFrenzied = n;
    }
    else if (t.code == 2)
    {
        n->next = manager.lastFrenzied->next;
        Manager.lastFrenzied->next = n;
    }
    else if (t.code == 3)
    {
        n->next = NULL;
        Manager.lastRegular = Manager.lastRegular->next = n;
    }
}
```

ⓒ
```
if (manager.head ≠ NULL)
{ Task t = manager.head->data;
  Manager.head = Manager.firstFrenzied = manager.head->next;
  return t
}
else
{ return NULL;
}
```

④
```
Node *Operator[] (const Matrix &M, int x)   //complexity is O(1)
{ return M.arrayX[x];
}
```

```
Node operator[] (Node *head, int y) // complexity is O(n)
{
    while (head->y < y)
    { head = head->next;
    }
    return (head->y == y)? head : Node (0);
}

Matrix addMatrix( const Matrix& A, const Matrix &B)
{ Matrix c(A.m, A.n);
  for (int i=0; i< c.m; i++)      // O(n)
   { for (int j=0; j<c.n; j++)    // O(n^2)
    {
      Node n = A[i][j] + B[i][j]; // O(n^3)
      c.insert (n,i,j);
    }
   }
  return c;
}
```

complexity is $O(n^3)$

② function (List L1, List L2)

```
{ List L3;
  for(int k=1; k≤ L1.size; k++)
  { Node* ck = L3.insertEnd (0);
    for (int i= 1; i≤k; i++)
    { Node* a = L1.start;
      for(int j=1; j≤i; j++)
      { a=a->next;
      }
      Node *b = L2.start;
      for(int j=1; j≤k- i+1; j++)
       {
        b=b->next;
       }
      ck->data += a->data * b->data;
    }
  }
  return L3;
}
```

complexity is $O(n^3)$

① Merge $(L_1, L_2)$

```
{
  if head(L1) == NULL then
  { head(L1) = head(L2);
    head(L2) = NULL;
  }
  x = head(L1);
  y = head(L2);
  x2 = NULL;
  while(x != NULL && y != NULL) do
  {
    if Key(x) < Key(y)
    { x2 = x;
        x = Next(x);
    }
    else
    { z = List - Delete - First(L2);
      y = head(L2);
      If (x2 != NULL)
      {
        List - Insert - after (L1, z, x2);
      } \* Insert z after x2 *\
    else
    { List - Insert - first (L1, z);
    }
    x2 = z;
  }
}
```

```
If (y != NULL) do
  {
    Next(x2) = y
    Head [L2] = NULL;
  }
Return (L1);
}
```