INFORMATION SYSTEMS

Programming I Exam Notes

```
double sqrt-bin (double x)
{ double y = 1.0;
  while (y*y < x){
    y = 2.0*y
  }
  return binary-sqrt (0, y, 0.01, x);
} //end program

double binary-sqrt(double low, double high, double tol, double n){
  double mid;
  while (low+tol <high){
    mid = (low+high)/2.0;
    if (mid*mid - n ≤ 0.0){
      low=mid
    } else {
      high=mid;
    }
  return low;
} //end program
```

Binary Search

```
public void search (int low, int high, String key){
  assert (is_ordered(low, high));
  int i = low-1;
  int j = high+1;
  while (i+1 != j){
    mid= (i+j)/2;
    if ( le (arr[mid], key)){
      i=mid
    } else {
      j=mid;
    } index=i;
  found = (low ≤ index) && key.equals (arr[index]);
}
```

INFORMATION SYSTEMS

2.

```
public void insert (String Str, int pos, int low, int high) {
  for (int k=high+1 ; k >pos; k--) {
      arr [k]  = arr[k-1];
  }
   arr [pos] = Str
} //end

public void Insert_Sort (int low, int high) {
  for (int k= low+1; k <= high; k++) {
      Search (low, k-1, arr[k]);
      Insert (arr[k], index+1, low, k-1);
  }
}//end
```

INFORMATION SYSTEMS

3

Programming I Exam Notes

Matrix Saddle point proof:

Any two Saddle points in a zero sum matrix must have the same value

Proof: Suppose a and b are two saddle points in a matrix. If they lie in the same row, then since a is the smallest entry in its row $a \leq b$; but since b is also the smallest entry in its row $b \leq a$. So $a = b$

Similarly, if they lie in the same column, both are the largest entry in the column, so both $a \geq b$ and $b \geq a$ So $a = b$

If they lie in a different row and column, then they form opposite corners of a rectangle in the matrix; let c and d be the other two corners their relative position in the matrix look like:

a . . c
d . . b

Since a is the smallest entry in its row and b the largest in its column, we have $a \leq c \leq b$. But then b is the smallest entry in its row and a the largest in its column so $b \leq d \leq a$. But since the same number ~~list~~ of putting these inequalities together: $a \leq c \leq d \leq a$. But since the same number lies at both ends of inequality, all 4 numbers must in fact be equal. This shows that not only does $a = b$, but that c and d are also saddle points

Matrix (class)

```
public Basic_Matrix transpose(){
    Basic_Matrix result = new Basic_Matrix(cols, rows);
    for (int i=0; i<cols; i++){
        for (int j=0; j<rows; j++){
            result.mat[i][j] = mat[j][i];
        }
    }
    return result;
}

int min_index(double arr[]){
    int result=0;
    double min=arr[0];
    for(int k=1; k<arr.length; k++){
        if(arr[k]<min){
            min=arr[k];
            result=k;
        }
    }
    return result;
}

int max_index(double arr[]){
    ''
    max=arr[0]
    ''
    if(arr[k]>max{
    max=arr[k];
    result=k;
    }
    ''
    ''

double[] min_row(){
    double[]result;
    double [] row_k;
    result = new double [rows];
    for(int k=0; k<rows; k++){
        row_k = mat[k];
        result[k] = row_k[min_index(row_k)];
    }
    return result;
}

double[] max_row(){
    ''
    ''
    ''
    ''
    ''
    result[k]=row_k[max_index(row_k)]
    ''
    ''
```

5

Programming 1 Exam Model

```
void one_saddle (Basic_Matrix m){
    double [] mins,
    double [] maxs;
    int mn, mx;
    Basic matrix mt;

    mins = m.min_row();
    mt = m.transpose();
    maxes = mt.max_row();
    mn = m.max_index(mins);
    mx = m.min_index(maxes);
    if (mins[mn] == maxes[mx]){
        print_Saddle(m.mat, mn, mx);
    } else {
        "No saddle point"
    }
}
void all_Saddle (Basic_Matrix m){
    double [] min, [] maxes;
    boolean found = false;
    Basic matrix mt;
    mins = m.min_row();
    mt = m.transp
    maxes = mt.max_row();
    for (int i=0; i<m.rows; i++){
        for (int k=0; k<m.cols; k++){
            if (mins[i] = maxs[j]){
                found = true;
                print_Saddle (m.mat, i, j);
            }
        }
    }
    if (!found){ print "no" }
}
```

INFORMATION SYSTEM

6

QuickSort

```
void qsort (int left, int right){
    String pivot;
    int i,j;

    pivot = arr[(left+right)/2];
    partition (left, right, pivot);
    i=L;
    j=R;
    if(left<j) { qSort(left,j); }
    if (i<right) { qsort (i,right); }
}
void partition (int LO, int RO, String p){
    L=LO;
    R=RO;
    while (L<=R){
        while (lt(arr[L], p)){    //left Scan
            comps++;
            L=L+1;
        }
        while (lt(p, arr[R])){ //right Scan
          comps++;
          R=R-1;
        }
        comps++;
        if (L<=R){
          exchange(L,R);
          exch++;
          L=L+1;
          R=R-1;
        }
    }
} //end partition
```

7

Programing 1 Exam Notes.

```
void Find (int k, int left, int right) {        //kth smallest element
  int i = left;
  int j = right;
  String pivot;
  while (i < j) {
      pivot = arr[k];
      partition (i, j, pivot);
      assert (R < L);
      if (R < k) {
        i = L;
      }
      if (k < L) {
        j = R;
      }
    }
}
void exchange (int i, int j) {
  String it;
  it = arr[i];
  arr[i] = arr[j];
  arr[j] = it;
}
```

L R k

if R < k
i = L

if R > k
j = R.

8.

Merge Sort

```
Linked_List merge (Linked_List left, Linked_List right){
    //left and right are ordered
    String x, y;
    Linked_List xs = left;
    Linked_List ys = right;
    Linked_List result;

    if (xs.is_empty()){
      result = ys;
    } else if (ys.isempty()){
      result =xs;
    } else {
      x = xs.first_item();
      y = ys.first_item();
      if ( le (x,y)){
          xs.removefirst();
          result = merge (xs, ys);
          result.prepend(x);
      } else {
            ys.remove first();
            result = merge(xs, ys);
            result.prepend (y);
      }
    }
return result;
}
```

9

Programming 1 Exam Model

```
Linked_List merge_sort (Linked_List ys){
  Linked_List xs, result;
  Linked_List left, right;
  if (ys.size <2){
      result = ys;
  }else{
      xs = new Linked_List();
      int h = 1;
      while (h <= (ys.size)/2 ){
        xs.prepend (ys.first item);
        ys.removeFirst();
        h = h+1;
      }
      left = mergeSort (xs);
      right = mergeSort (ys);
      result = merge (left, right);
  }
 return result;
}
```

Programming 1.

Binary square root // Binary search

```
double binary_sqrt(double low, double high, double tol, double n){
    double mid;
    while ( low + tol < high){
        mid = (low + high) /2;
        if (mid * mid - n <= 0){
            low = mid;
        else{
            high = mid;
        }
        return low;
}// finish func
```

Actual Binary search method

```
public void search (int low, int high, String str){
    assert (is_ordered (low, high));
    int i = low -1;
    int j = high +1;
    while (i + 1 != j){
        mid = (i+j)/2;
        if ( le (arr[mid], key) ){
            i = mid;
        }else{
            j = mid;
        }
    }
    index = i;
    found = (low <= index) && (key. equals (arr[index]));
}
```

Insert Sort method continued from binary search

```
public void insert(String str, int pos, int low, int high){
    for(int k = high+1; k > pos; k--){
        arr[k] = arr[k-1];
    } arr[pos] = str
}
```

Insert Sort extension of BS and insert

```
public void insert_Srt(int low, int high){
    for(int k = low+1; k <= high; k++){
        Search(low, k-1, arr[k]);
        insert(arr[k], index+1, low, k-1);
    }
}
```

Programmy 1.

QUICK SORT METHOD

```
void partition (int LO, int RO, String p)
{
  L=LO;
  R=RO;
  while (L<=R){
    //left scan
    while ( lessthan(arr[L],p){
        comp=comps+1
        L=L+1
      }
    while ( lessthan(p, arr[R]){
        R=R-1
      }
    if (L<=R){
        exchange(L,R);
        L=L+1
        R=R+1
      }
  }
} //end partition

Find Method finds kth smallest element in [left,... right]
void find (int, int left, int right){
    int i=left;
    int j=right;
    String pivot
    while (i<j){
      pivot=arr[k];
      partition (i, j, pivot);
      assert (R<L);
        if (R<k){ i=L; }
        if (k<L){ j=R;
      }
  } // end while
```

Quicksort method:

```
void qSort (int left, int right) {
    String pivot;
    int i, j;
      pivot = arr[(left+right)/2];
      partion (left, right, pivot);
      i=L;
      j=R;
      if (left <j){ qSort (left, j)}
      if (i <right) { qsort(i, right)
```