Item.h:

```
Key KEYget(ITEM item);
int KEYeq(Key k, Key m);
int KEYless(Key k, Key m);
int KEYgreater(Key k, Key m);
int KEYcompare(Key k, Key m);
Key KEYscan();
ITEM ITEMsetvoid();
ITEM ITEMrand ();
int ITEMcheckvoid(ITEM item);
ITEM ITEMscan();
ITEM ITEMfscan(FILE *file);
void ITEMfree(ITEM item);
ITEM ITEMchange();
```

List.h:

```
link NEWnode(ITEM item, link x);
link LISTinshead(link x, ITEM item);
link LISTinstail(link x, ITEM item);
ITEM LISTsearch(link x, Key k);
link LISTdelhead(link x);
ITEM LISTextrheadP(link *x);
link LISTdelkey(link x, Key k);
link LISTdelkeyR(link x, Key k);
ITEM LISTextrkeyP(link *x, Key k);
link LISTsortins(link x, ITEM item);
ITEM LISTsortsearch(link x, Key k);
link LISTsortdel(link x, Key k);
void LISTshow(link x);
void LISTfree(link x);
```

ST.h:

```
ST STinit(int N);
void STdisplay(ST tabella);
int STsize(int N);
int STinsert(ST tabella, ITEM item);
int STcount(ST tabella);
int STempty(ST tabella);
int STselect(ST tabella, int r);
int STgetindex(ST tabella, ITEM item);
ITEM STsearch(ST tabella, Key k);
Key STsearchByIndex (ST st, int id);
void STdelete(ST tabella, Key k);
void STfree(ST tabella);
void STdisplay (ST st);
int hashU(char *v, int M);
int hash (Key k, int M);
int full((ST tabella, int i);
```

Queue.h

```
QUEUE QUEUEinit(int N);
int QUEUEempty(QUEUE queue);
void QUEUEput(QUEUE queue, ITEM item);
ITEM QUEUEget(QUEUE queue);
```

Stack.h

```
STACK STACKinit(int N);
int STACKempty(STACK stack);
void STACKpush(STACK stack, ITEM item);
ITEM STACKpop(STACK stack);
```

BST.h:

```
BST BSTinit();
void BSTfree (BST bst);
int BSTcount (BST bst);
int BSTempty (BST bst);
Item BSTsearch(BST bst, Key k);
Item BSTmin(BST bst);
Item BSTmax(BST bst);
void BSTinsert_leafR(BST bst, Item x);
void BSTinsert_leafI(BST bst, Item x);
void BSTinsert_root(BST bst, Item x);
void BSTvisit (BST bst, int strategy);
link rotR(link h);
link rotL(link h);
link partR (link h, int r);
void BSTdelete (BST bst, Key k);
Item BSTselect (BST bst, int r);
Item BSTsucc (BST bst, Key k);
Item BSTpred (BST bst, Key k);
```

IBST.h:

```
void IBSTinit(IBST ibst);
void IBSTfree (IBST ibst);
void BSTinsert (IBST ibst, Item x);
void IBSTdelete (IBST ibst, Item x);
Item IBSTsearch (IBST ibst, Item x);
Int IBSTcount (IBST ibst);
int IBSTempty (IBST ibst);
void IBSTvisit (IBST ibst, int strategy);
```

PQ.h:

```
PQ PQinit(int N);
void PQfree (PQ pq);
int PQempty(PQ pq);
void PQinsert(PQ pq, ITEM item);
ITEM PQextractMax(PQ pq);
ITEM PQshowMax(PQ pq);
void PQdisplay(PQ pq);
int PQsize(PQ pq);
void PQchange(PQ pq, ITEM x);
void PQchange(PQ pq, int pos, ITEM x);
```

Heap.h:

```
HEAP HEAPinit(int N);
void HEAPfill(HEAP heap, ITEM item);
void HEAPsort(HEAP heap);
void HEAPdisplay(HEAP heap);
void HEAPfree(HEAP heap);
void HEAPify(HEAP heap, int i);
void HEAPbuild(HEAP heap);
int PARENT(int i);
int RIGHT(int i);
int LEFT(int i);
```

Graph.h:

```
Graph GRAPHinit(int V);
void GRAPHfree(Graph G);
Graph GRAPHload(FILE *fin);
void GRAPHstore(Graph G, FILE *fout);
void GRAPHgetIndex(Graph G, char*label);
void GRAPHinsertE(Graph G, int id1, int id2, int wt);
void GRAPHremoveE(Graph G, int id1, int id2);
void GRAPHshow(Graph G);
void GRAPHedges(Graph G, Edge *a);
void insertE(Graph G, Edge e);
void removeE(Graph G, Edge e);
int randV(Graph G);
Graph GRAPHrand1(Graph G, int V, int E);
Graph GRAPHrand2(Graph G, int V, int E);
int GRAPHpath(Graph G, int id1, int id2);
void GRAPHpathH (Graph G, int id1, int id2);
void GRAPHbfs(Graph G, int id);
void GRAPHdfs(Graph G, int id);
int GRAPHscc(Graph G);
int GRAPHcc(Graph G);
Graph reverse(Graph G);
void GRAPHmstK(Graph G);
void GRAPHmstP(Graph G);
void GRAPHspD(Graph G, int id);
void GRAPHspBF(Graph G, int id);
```

Edge.h:

```
Edge EDGEalloc(int v, int w, int peso);
Edge EDGEcreate(int v, int w, int peso);
```

Search.h

```
ITEM LinearSearch(ITEM * item, int l, int r, int k);
ITEM BinarySearch(ITEM * item, int l, int r, int k);
ITEM BinarySearchR(ITEM * item, int l, int r, int k);
```

IterativeSort.h

```
void BubbleSort(ITEM *item, int l, int r);
void OptBubbleSort(Item A[], int N);
void InsertionSort(ITEM *item, int l, int r);
void ShellSort(ITEM *item, int l, int r);
void SelectionSort(ITEM *item, int l, int r);
void CountingSort(ITEM *item, int l, int r, int k);
```

RecursiveSort.h

```
void QuickSort (ITEM *item, int N);
void QuickSortR(ITEM *item, int l, int r);
int partition(ITEM *item, int l, int r);
void MergeSort (ITEM *A, int N);
void MergeSortR(ITEM *A, ITEM * B, int l, int r);
void Merge(ITEM *A, ITEM *B, int l, int q, int r);
void BottomUpMergeSort (ITEM *A, int N);
```