國立成功大學 108 學年度碩士班招生考試試題

系　　所：工程科學系

考試科目：資料結構

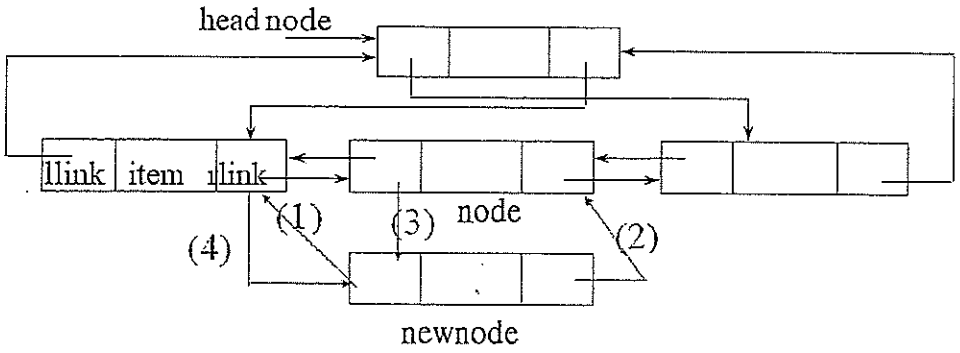
考試日期：0224，節次：1

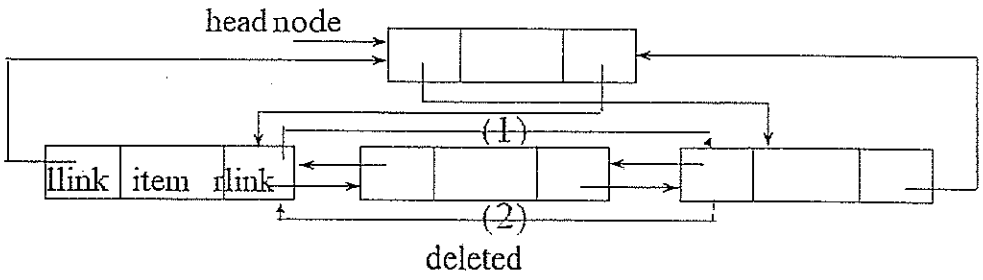
※ 考生請注意：本試題不可使用計算機。　請於答案卷(卡)作答，於本試題紙上作答者，不予計分。

1. Complete the functions of inserting a newnode into a doubly linked circular list and deleting one node from a doubly linked circular list. (12%)

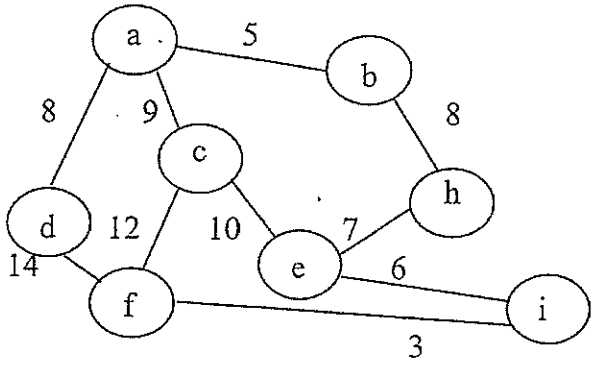
```
void dinsert(node_pointer node, node_pointer newnode)
{
    (1);
    (2);
    (3);
    (4);
}

void ddelete(node_pointer node, node_pointer deleted)
{
    if (node==deleted) printf("Deletion of head node not permitted.\n");
    else {
        (1)
        (2)
            free(deleted);
    }
}
```





2. Please write down every sequence of each step for finding the Minimum Cost Spanning Tree using Kruskal and Prim Algorithms. (8%)



3. For the sequence: 33, 32, 23, 7, 5, 100, 46, 67, 20. Please write down every sequence of each step while applying Insertion Sort, Quick Sort, Interactive Merge Sort, Recursive Merge Sort, MAX Heap Sort and LSD Radix Sort (5%), then complete the Sort algorithms parts indicated by XXX (2%) (80%)

Heap Sort

```
# define MAX-SIZE 1000/* maximum size of   list plus one */
typedef struct {
                    int key;
                    /* other fields */
                    } element;
element list[MAX_SIZE];

void adjust(element list[], int root, int n)
{
  int child, rootkey;
  element temp;
  temp=list[root];
  rootkey=list[root].key;
  child=2*root;
  while (child <= n) {
    if ((child < n) &&
        (list[child].key < list[child+1].key))
            child++;
```

```
        if (rootkey > list[child].key) break;
        else {
            XXXX_1;
            XXXX_2;
        }
    }
    list[child/2] = temp;
}
void heapsort(element list[], int n)
{
    int i, j;
    element temp;
    for (i=n/2; i>0; i--)
        XXXX_3;
    for (i=n-1; i>0; i--) {
        SWAP(list[1], list[i+1], temp);
        XXXX_4;
    }
}
```

Quick Sort

```
void quicksort(element list[], int left, int right)
{
  int pivot, i, j;
  element temp;
  if (left < right) {
    i = left; j = right+1;
    pivot = list[left].key;
    do {
      do i++; while (XXXX_5);
      do j--; while (XXXX_6);
      if (i < j) SWAP(list[i], list[j], temp);
    } while (i < j);
    SWAP(list[left], list[j], temp);
    quicksort(XXXX_7);
    quicksort(XXXX_8);
```

```
    }
}
```

Interactive Merge Sort

```
void merge(element list[], element sorted[], int i, int m, int n)
{
   int j, k, t;
   j = m+1;
   k = i;
   while (i<=m && j<=n) {
      if (list[i].key<=list[j].key)
         XXXX_9;
      else XXXX_10;
   }
   if (i>m) for (XXXX_11)
      sorted[k+t-j]= list[t];
   else for (XXXX_12)
   sorted[k+t-i] = list[t];
}

void merge_pass(element list[], element sorted[],int n, int length)
{
   int i, j;
   for (i=0; i<n-2*length; i+=2*length)
      merge(XXXX_13);
   if (i+length<n)
      merge(XXXX_14);
   else
      for (j=i; j<n; j++) sorted[j]= list[j];
}

void merge_sort(element list[], int n)
{
      int length=1;
      element extra[MAX_SIZE];
      while (length<n) {
```

```
            merge_pass(XXXX_15);
            length *= 2;
            merge_pass(XXXX_16);
            length *= 2;
        }
}
```

Recursive Merge Sort

```
int rmerge(element list[], int lower, int upper)
{
   int middle;
   if (lower >= upper) return lower;
   else {
      middle = XXXX_17;
      return listmerge(XXXX_18);
   }
}

int listmerge(element list[], int first, int second)
{
   int start=n;
   while (first!=-1 && second!=-1) {
      if (list[first].key<=list[second].key) {
      /* key in first list is lower, link this
           element to start and change start to
           point to first */
         XXXX_19;
         start = first;
         XXXX_20;
      }
      else {
   /* key in second list is lower, link this
        element into the partially sorted list */
         XXXX_21;
         start = second;
         XXXX_22;
```

```
        }
    }
    if (first==-1)
        list[start].link = second;
    else
        list[start].link = first;
    return list[n].link;
}
```

LSD Radix Sort

```
#define MAX_DIGIT 3
#define RADIX_SIZE 10
typedef struct list_node *list_pointer;
typedef struct list_node {
        int key[MAX_DIGIT];
        list_pointer link;
}
list_pointer radix_sort(list_pointer ptr)
{
    list_pointer front[RADIX_SIZE],
                 rear[RADIX_SIZE];
    int i, j, digit;
    for (i=MAX_DIGIT-1; i>=0; i--) {
        for (j=0; j<RADIX_SIZE; j++)
            front[j]=read[j]=NULL;
        while (ptr) {
            digit=ptr->key[I];
            if (!front[digit]) XXXX_23;
            else rear[digit]->link=ptr;
            XXXX_24;
            ptr=ptr->link;
        }
        /* reestablish the linked list for the next pass */
        ptr= NULL;
        for (j=RADIX_SIZE-1; j>=0; j--)
            if (front[j]) {
```

```
            XXXX_25;
            ptr=front[j];
        }
    }
    return ptr;
}
```