```
/* --- sort 2.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct record {
  char name[10];
  int point;
  struct record *next;
};

void insert_list(char *name, int x);
struct record *head = NULL;

int main(int argc, char *argv[])
{
  FILE *fp;
  int x;
  char name[10], buf[256];
  struct record *p;

  if (argc != 2) {
    printf("missing file argument¥n");
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    printf("can't open %s¥n", argv[1]);
    return 1;
  }

  while (fgets(buf, sizeof(buf), fp) != NULL) {
    sscanf(buf, "%s %d", name, &x);
    insert_list(name, x);
  }

  fclose(fp);

  for(p = head; p != NULL; p = p->next)
    printf("%s %d¥n", p->name, p->point);

  return 0;
}
```

```
void insert_list(char *name, int x)
{
  struct record *p, *q, *t;

  t = (struct record *) malloc(sizeof(struct record));
  if (t == NULL) {
    printf("Out of memory¥n");
    exit(1);
  }
  strcpy(t->name, name);
  t->point = x;

  q = NULL;
  for(p = head; p != NULL; p = p->next){
    if(p->point >= x) break;
    q = p;
  }

  if(q != NULL)
    q->next = t;
  else
    head = t;
  t->next = p;
}
```

```
/* sort3 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
  char name[10];
  int point;
  struct node *left;
  struct node *right;
};

void insert_data(char *name, int x, struct node **pp);
void printtree(struct node *p);

struct node *root = NULL;

int main(int argc, char *argv[])
{
  FILE *fp;
  int x;
  char name[10], buf[256];
  struct record *p;

  if (argc != 2) {
    printf("missing file argument¥n");
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    printf("can't open %s¥n", argv[1]);
    return 1;
  }

  while (fgets(buf, sizeof(buf), fp) != NULL) {
    sscanf(buf, "%s %d", name, &x);
    insert_data(name, x, &root);
  }

  fclose(fp);

  printtree(root);

  return 0;
}
```

```
void insert_data(char *name, int x, struct node **pp)
{
  struct node *p, *t;

  p = *pp;
  if(p == NULL){
    t = (struct node *) malloc(sizeof(struct node));
    if (t == NULL) {
      printf("Out of memory¥n");
      exit(1);
    }
    strcpy(t->name, name);
    t->point = x;
    *pp = t;
    return;
  }

  if(x <= p->point)
    insert_data(name,x,&p->left);
  else
    insert_data(name,x,&p->right);
}

void printtree(struct node *p)
{
  if(p == NULL) return;
  printtree(p->left);
  printf("%s %d¥n", p->name, p->point);
  printtree(p->right);
}
```

```
/* sort4 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
  char name[10];
  int point;
  struct node *left;
  struct node *right;
};

struct node *insert_data(char *name, int x, struct node *p);
void printtree(struct node *p);

struct node *root = NULL;

int main(int argc, char *argv[])
{
  FILE *fp;
  int x;
  char name[10], buf[256];
  struct record *p;

  if (argc != 2) {
    printf("missing file argument¥n");
    return 1;
  }

  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    printf("can't open %s¥n", argv[1]);
    return 1;
  }

  while (fgets(buf, sizeof(buf), fp) != NULL) {
    sscanf(buf, "%s %d", name, &x);
    root = insert_data(name, x, root);
  }

  fclose(fp);

  printtree(root);

  return 0;
}
```

```
struct node *insert_data(char *name, int x, struct node *p)
{
  if(p == NULL){
    p = (struct node *) malloc(sizeof(struct node));
    if (p == NULL) {
      printf("Out of memory¥n");
      exit(1);
    }
    strcpy(p->name, name);
    p->point = x;
    return p;
  }

  if(x <= p->point)
    p->left = insert_data(name,x,p->left);
  else
    p->right = insert_data(name,x,p->right);
}

void printtree(struct node *p)
{
  if(p == NULL) return;
  printtree(p->left);
  printf("%s %d¥n", p->name, p->point);
  printtree(p->right);
}
```