Fall 2013, CS288 Test 2, 1-2:15 pm, Fri, 10/25/2013, CKB204

Name:

The exam assumes 32-bit Linux machines. The exam has 5 pages. Make sure you have all the pages. Do not take any page(s) with you. Any missing page(s) will result in failure in the exam. This exam is closed book close notes. Do not exchange anything during the exam. No questions will be answered during the exam. If you are in doubt, briefly state your assumptions below, including typos if any. Questions 1 to 15 are worth 3 points each.

ANSWERS

I have read and understood all of the instructions above. On my honor, I pledge that I have not violated the provisions of the NJIT Academic Honor Code.

Signature: ____________________ **Date:** __________

Questions 1-3: Assume radix sorting of 1024 unsigned integers on a 32-bit machine with 4 passes (rounds). The integers are initially stored in lst[1024] and the sorted integers will be available in lst at the end of sorting. int buf[1024] is available as working space.

1. What is the number of buckets?

   1<<(b/p)   225

2. The bit mask in *hexadecimal* is?

   (1<<(b/p))-1   0xFF

3. Find the number of data assignments for correcting the result after 4 passes are completed. For example, moving lst[i] to buf[j], or buf[j]=lst[i]; is a data assignment.

   0, b/c sorted contains only positive numbers

4. For floating point radix sort, assuming exactly half (512 floats) is negative, what is the number of data assignments for correcting the result?

   2n

5. Given `float f;` write a C statement to access the binary equivalent of `f`:

   *(unsigned float *) (&f)

6. In class, it was demonstrated through realtime execution that selection sort is super slow, merge sort is acceptable while radix sort is fast. Explain their performance in a sentence with not more than 20 words. You may use big O notation to simplify your answer.
   Selection sort = O(n^2); Merge sort = O(n log n); Radix sort = O(n)

7. Given `char *str = "?a???b,,,#c";` what would `strtok(str, "?");` return?
   "a"

8. Continuing Problem 7 write fill in the blank such that strtok() will return `??b`.

   `strtok(`______NULL, ","______`);`

9. Continuing Problem 8, what would `strtok(NULL, "#,");` return?
   "c"

For 10-15 on the 15-puzzle state-space search, consider open=(x,y) and succ=(p,q,r). Assuming the first one in open is always picked for search,

10. find open after merging with succ using breadth-first search strategy.
    open = (x, y, p, q, r)

11. find open after merging with succ using depth-first search strategy.
    open = (p, q, r, x, y)

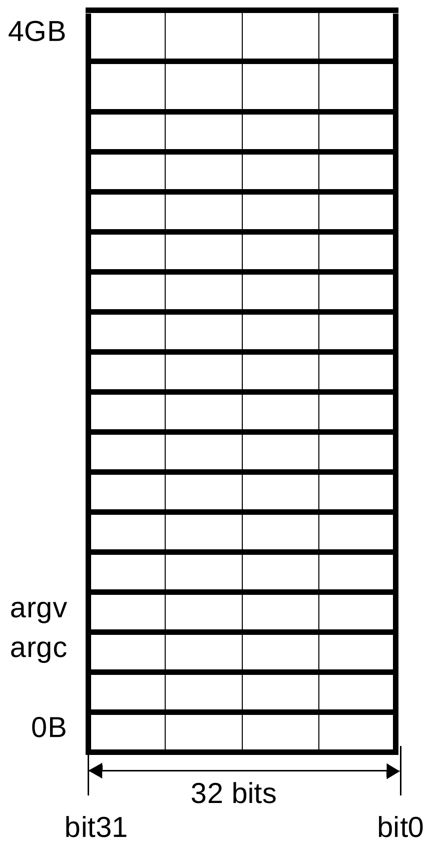
12. find open after merging with succ using best-first search strategy.
    open = (p, x, q, y, r)

13. What is the branching factor for the 15-Puzzle problem?
    [2(4)+3(8)+4(4)]/16 = 3   3

14. Given `struct x { int a,b; struct x *p,*q; };` what would `sizeof(struct x)` return?
    16 bytes

15. Given `struct x { int **a,**b; struct x **p,**q; };` what would `sizeof(struct x)` return?
    **a -> 4
    **b -> 4   16 bytes
    **p -> 4
    **q -> 4

Problem 16 (comand line arugments - 15 points) At the command line prompt, you type “xyz 123 abc” and hit enter, where xyz is your C executable while 123 and abc are parameters. Show the contents of memory for argv, intermediate pointers, and the parameters in the table on the right. Use arrows to indicate the relationship between them.



Problem 17 (Unsigned integer radix sort - 20 points): Write a C program for sorting 32-bit unsigned integers using radix sort with a group of 4 bits. Use the variables listed below. Assume lst is initialized with n numbers.

```
#define N 1048576
#define BIN 256
#define MAXBIT 32
#define LST 1
#define BUF 0
```

```
int n,group,bin;
int flag; /* to show which one holds numbers: lst or buf */
int lst[N],buf[N];
int count[BIN], map[BIN], tmap[BIN];
```

```
int main(int argc, char **argv){
    int i;
    flag = LST;
    initialize(); /* initialize lst with n random floats */
    for (i=0;i<MAXBIT;i=i+group) radix_sort(i); /* move lst to buf or buf to lst depending on the iteration number */
    correct(); /* sorted numbers must be in lst */
}
void radix_sort(int idx) {
    int i,j,k,mask; /* initialize mask for lifting the 4 least significant bits. */
    int *src_p,*dst_p; /* cast lst and buf to int pointers to treat lst/buf as int's */
    /* set src_p and dst_p*/  if ( flag = LST) {               else {
                                      src_p = lst                  src_p = buf
                                      dst_p = buf                  dst_p = lst
                                      flag = BUF                   flag = LST
                              }                                }
    /* count */  mask = 0xFF                 for (i = 0; i < N; i++) {
                 for (i = 0; i < bin; i++) {          count [(src_p[i] >> idx) & mask]++
                         count[i]=0           }
                         map[i] = 0
                 }

    /* map */    for (i = 1; i < bin; i++) {
                         map[i] = map[i-1] + count [i-1]
                 }

    /* move */   for (i = 0; i < N; i++) {
                         dest_p [map [(src_p[i] >> idx) & mask]++] = src_p

}

void correct() {  for (i =0; i < N; i++) {                    for (i = 0; i < N; i++ {
                          if ((i + n) < N) {                          lst[i] = buf[i]
                                  buf[i] = lst [i + n]        }
                          } else {
                                  buf [i] = lst [i - (N-n)]
                  }
}
```

Problem 18 (Linked list - 20 points): Write two C functions append() and len() where
1. append() creates a struct clip, sets the value passed as parameter to number, and adds the newly created struct to the end of the list. *head* points to the first node in the list. Your function must be able to handle any number of nodes, including none in the beginning.
2. length() returns the number of clips in the list pointed by head.

```
#include <stdio.h>
void append(); int length();
struct node { int number; struct node *next; } *head;
void main() {
  struct node *head; int n;
  append(&head,1);
  n = length(head);
}

void append(____struct node **hp int data____) {
  struct clip *cp,*tp; /* for new clip, traversing as needed */
    cp = (sn*)malloc(size of (sn))
    cp -> number = data
    cp -> next = NULL
    if (*hp == NULL) {
            *hp = cp;
            return;
    }
    tp = *hp
    while (tp -> next != NULL) {
            tp = tp-> next;
    }
    tp -> next = cp;
}

int length(____struct hp____) {
    int i = 0;
    while (hp != NULL) {
            hp = hp -> next
            i++
    }
    return i;
}
```