MC140: lecture #22

today's topic:

selection sort

selection sort.

- selection sort uses an auxiliary array
- it copies elements, one at a time, from the array being sorted into the auxiliary array
- the auxiliary array is always kept in sorted order. entries are *selected* from the array being sorted, one at a time in sorted order, and inserted sequentially into the auxiliary array

selection sort, 2.

- hey, this sounds just like insertion sort!
- but the difference is that elements are *removed* from the array being sorted, *in sorted order*
- so smallest element in the array being sorted is removed and placed in the position of the last element in the auxiliary array

selection sort, 3.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

/* define constant */
#define NUM_DICE 10

/* function prototypes */
int roll_die();
void roll_dice( int dice[],int size );
void print_dice( int dice[],int size );
void sort_dice( int *dice,int size );
void swap( int *a,int *b );
```

```
int main( void ) {

   int i, j;
   int dice[NUM_DICE];

   /* initialize random seed */
   srand( time ( NULL ));

   /* fill the dice array with
      random numbers */
   roll_dice( dice,NUM_DICE );

   /* print the dice */
   printf( "messy dice: " );
   print_dice( dice,NUM_DICE );

   /* sort the dice and print
      them again */
   sort_dice( dice,NUM_DICE );
   printf( "nice dice: " );
   print_dice( dice,NUM_DICE );

   return( 0 );
} /* end of main() */
```


selection sort, 4.

```
int roll_die() {
  return(( rand() % 6 ) + 1 );
} /* end of roll_die() */

void roll_dice( int dice[], int size ) {
  int i;
  for ( i=0; i<size; i++ ) {
    dice[i] = roll_die();
  } /* end for i */
} /* end of roll_dice() */

void print_dice( int dice[], int size ) {
  int i;
  for ( i=0; i<size; i++ ) {
    printf( "%d ",dice[i] );
  } /* end for i */
  printf( "\n" );
} /* end of print_dice() */
```


selection sort, 5.

```
/* this function returns the index of the smallest element
   in the dice array */
int smallest( int dice[], int size ) {
  int i;    /* loop control variable */
  int min;  /* value of smallest element in dice array */
  int minx; /* index of smallest element in dice array */
  min = dice[0];
  minx = 0;
  for ( i=1; i<size; i++ ) {
    if ( dice[i] < min ) {
      min = dice[i];
      minx = i;
    } /* end if */
  } /* end for i */
  return( minx );
} /* end of smallest() */
```

selection sort, 6.

```
void sort_dice( int dice[], int size ) {
   int aux[NUM_DICE], auxsize = 0;
   int i, j;
   for ( i=0; i<size; i++ ) {
      j = smallest( dice,size );
      aux[auxsize] = dice[j];
      auxsize++;
      dice[j] = 9999; /* set it to a large (out of range)
                         value so it won't be mistaken for
                         the smallest entry */
   } /* end for j */
   for ( i=0; i<size; i++ ) {
         dice[i] = aux[i];
   } /* end for i */
} /* end of sort_dice() */
```


sample run of selection sort.

- *first, we insert 2 -- the smallest entry -- into aux.*

dice = | 6 | 4 | 5 | 2 | 3 | aux = | 2 | | | | |

- *then we replace the 2 with our large, out-of-range value, 9999.*
- *next, we insert 3, the smallest remaining entry, into aux.*

dice = | 6 | 4 | 5 | 9999 | 3 | aux = | 2 | 3 | | | |

- *then we replace the 3 with 9999.*
- *next, we insert 4, the smallest remaining entry, into aux.*

dice = | 6 | 4 | 5 | 9999 | 9999 | aux = | 2 | 3 | 4 | | |

- *then we replace the 4 with 9999.*
- *next, we insert 5, the smallest remaining entry, into aux.*

dice = | 6 | 9999 | 5 | 9999 | 9999 | aux = | 2 | 3 | 4 | 5 | |

sample run of selection sort,2.

(copied from end of last slide)

dice = | 6 | 9999 | 5 | 9999 | 9999 | aux = | 2 | 3 | 4 | 5 | |

- *then we replace the 5 with 9999.*
- *next, we insert 6, the smallest remaining entry, into aux.*

dice = | 6 | 9999 | 9999 | 9999 | 9999 | aux = | 2 | 3 | 4 | 5 | 6 |

- *then we replace the 6 with 9999.*

dice = | 9999 | 9999 | 9999 | 9999 | 9999 |

aux = | 2 | 3 | 4 | 5 | 6 |

- *fiinally, we copy the contents of aux back to dice.*

dice = | 2 | 3 | 4 | 5 | 6 | aux = | 2 | 3 | 4 | 5 | 6 |

selection sort comments.

- note that after each item is selected from the array being sorted and inserted into the auxiliary array, it is replaced in the array being sorted with a large number, in this case 9999.
- the idea is to substitute a value that is out of range of the rest of the items in the array, so that it will never be mistaken as the smallest item (and thus mistakenly selected for removal to the auxiliary array).
- sometimes this is not possible, if there is no notion of a large, out-of-range value.

selection sort comments, 2.

- it depends on the application.
- if there is no notion of a large, out-of-range value, then the items can really be removed from the array being sorted, by shifting the contents of the array and decreasing its size.
- this is a more complex algorithm, however, and we won't cover it this semester.

reading.

- the material covered today is not in the text book
- EXAM #3 will be on WED 11 APRIL