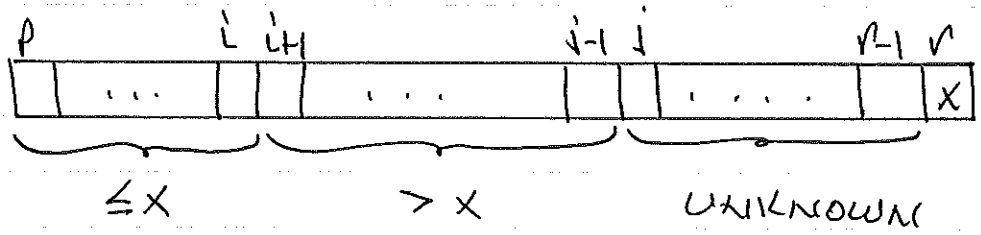
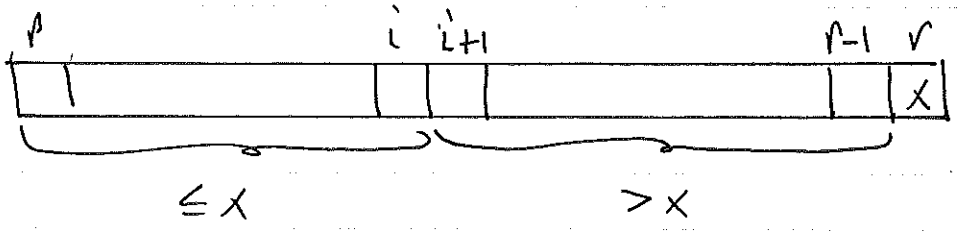
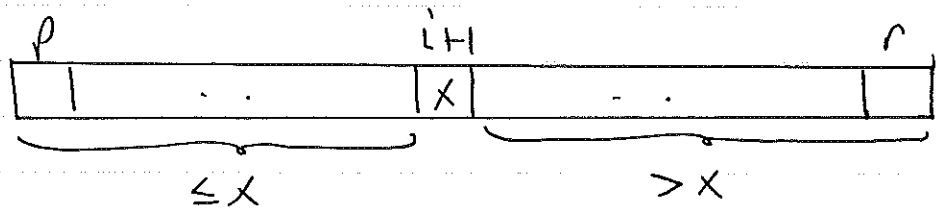
EXAMINING THE CODE FOR PARTITION WE SEE THAT THE FOR LOOP MAINTAINS THE INVARIANT



i.e. $A[p \ldots i] \leq x < A[i+1 \ldots j-1]$. THIS INVARIANT IS CERTAINLY TRUE BEFORE THE FIRST ITERATION, SINCE BOTH SUBSECTIONS ARE EMPTY. IT'S NOT HARD TO SEE THAT THE INVARIANT IS MAINTAINED FROM ONE ITERATION TO ANOTHER. WHEN $j = r$ WE HAVE



THE SINGLE SWAP $A[i+1] \leftrightarrow A[r]$ GIVES



THEN WE RETURN THE INDEX $i+1$.

QUICK SORT IS DUAL IN SOME SENSE TO Merge Sort. BOTH ARE DIVIDE AND CONQUER ALGORITHMS. Merge SORT DOES ITS REAL WORK IN THE RECOMBINE STEP, WHILE QUICKSORT DOES IT IN THE DIVIDE STEP.

THE RUNTIME OF QUICK SORT is

WORST CASE: #COMP $= \Theta(n^2)$
AVERAGE CASE: #COMP $= \Theta(n \lg n)$.

SO FAR WE HAVE

	WORST CASE	AVERAGE CASE
BUBBLE SORT	n^2	n^2
SELECTION SORT	n^2	n^2
Insertion SORT	n^2	n^2
Merge Sort	$n \lg n$	$n \lg n$
QUICK SORT	n^2	$n \lg n$

OFTEN THE PARTITION SUBROUTINE WILL BE RANDOMIZED IN ORDER TO ELICIT THE AVERAGE COST $\Theta(n \lg n)$ INSTEAD OF THE WORST CASE $\Theta(n^2)$.

WHAT is QUICK ABOUT QUICKSORT? THE CONSTANTS IMPLICIT IN THE Θ NOTATION ARE VERY SMALL, MAKING QUICK SORT VERY EFFICIENT, EVEN IN COMPARISON TO OTHER $\Theta(n \lg n)$ ALGORITHMS.

THEOREM
ANY COMPARISON BASED SORTING ALGORITHM DOES $\Omega(n \lg n)$ COMPARISONS ON ARRAYS OF LENGTH n IN WORST AND AVERAGE CASE.

TO SAY THAT AN ALGORITHM IS 'COMPARISON BASED' IS TO SAY THAT THE ONLY INFORMATION IT USES IS THE ORDER RELATION (i.e. $<, \leq, >, \geq$) ON THE ARRAY ELEMENTS.

WE WONT PROVE THIS THEOREM BUT ITS EASY TO SEE WHY IT SHOULD BE TRUE. THE SORTING PROBLEM ON AN ARRAY OF LENGTH n IS REALLY A SEARCHING PROBLEM ON A SPACE OF SIZE $n!$, i.e. WE MUST DETERMINE WHICH OF THE $n!$ PERMUTATIONS OF THE INPUT ARRAY REPRESENTS A REARRANGEMENT WHICH IS SORTED. EACH COMPARISON REDUCES THE SEARCH SPACE BY AT MOST HALF, SINCE EXACTLY HALF OF THE PERMUTATIONS OF A PUT A_i TO THE LEFT OF A_j, AND HALF PLACE A_j TO THE LEFT OF A_i.

THE NUMBER OF TIMES $n!$ CAN BE HALVED IS $\lg(n!)$, THUS

$$\#\text{Comparisons} = \Omega(\lg(n!)).$$

Lemma: $\lg(n!) = \Theta(n \lg n)$.

Proof

$$\lg(n!) = \sum_{i=1}^{n} \lg(i) \leq \sum_{i=1}^{n} \lg(n) = n \lg n.$$

Also

$$\lg(n!) = \sum_{i=1}^{n} \lg(i) \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \lg(i) \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \lg \lceil \tfrac{n}{2} \rceil$$

$$\therefore \lg(n!) \geq \left(n - \lceil \tfrac{n}{2} \rceil + 1\right) \lg \lceil \tfrac{n}{2} \rceil = \left(\lfloor \tfrac{n}{2} \rfloor + 1\right) \lg \lceil \tfrac{n}{2} \rceil$$

$$\geq \left(\tfrac{n}{2} - 1 + 1\right) \lg\left(\tfrac{n}{2}\right)$$

$$= \tfrac{n}{2}(\lg n - 1)$$

$$= \tfrac{1}{2} n \lg n - \tfrac{1}{2} n$$

BUT $n \geq 4 \Rightarrow \lg n \geq 2 \Rightarrow n \lg n \geq 2n \Rightarrow \tfrac{1}{4} n \lg n \geq \tfrac{1}{2} n$

$$\Rightarrow \tfrac{1}{2} n \lg n - \tfrac{1}{2} n \geq \tfrac{1}{4} n \lg n$$

$\therefore$ FOR ALL $n \geq 4$ WE HAVE

$$0 \leq \tfrac{1}{4} n \lg n \leq \lg(n!) \leq n \lg n$$

WHENCE $\lg(n!) = \Theta(n \lg n)$. ///

```
/*
*  Sort.c
*/

#include<stdio.h>
#include<stdlib.h>
#include<assert.h>

void swap(int* A, int i, int j){
   int temp;
   temp = A[i];
   A[i] = A[j];
   A[j] = temp;
}

void BubbleSort(int* A, int n){
   int i, j;
   for(j=n-1; j>0; j--){
      for(i=1; i<=j; i++){
         if( A[i]<A[i-1] ) swap(A, i, i-1);
      }
   }
}

void SelectionSort(int* A, int n){
   int i, j, imax;
   for(j=n-1; j>0; j--){
      imax = 0;
      for(i=1; i<=j; i++) if( A[imax]<A[i] ) imax = i;
      swap(A, imax, j);
   }
}

void InsertionSort(int* A, int n){
   int i, j, temp;
   for(j=1; j<n; j++){
      temp = A[j];
      i = j-1;
      while( i>=0 && temp<A[i] ){
         A[i+1] = A[i];
         i--;
      }
      A[i+1] = temp;
   }
}

void Merge(int* A, int p, int q, int r){
   int i, j, k, n1=q-p+1, n2=r-q;
   int* L = calloc(n1, sizeof(int));
   int* R = calloc(n2, sizeof(int));
   assert(L!=NULL && R!=NULL);

   for(i=0; i<n1; i++) L[i] = A[p+i];
   for(j=0; j<n2; j++) R[j] = A[q+j+1];
   i = 0; j = 0;
   for(k=p; k<=r; k++){
      if( i<n1 && j<n2 ){
         if( L[i]<R[j] ){ A[k] = L[i]; i++;}
         else{ A[k] = R[j]; j++;}
      }
      else if( i<n1 ){ A[k] = L[i]; i++;}
      else{ /* j<n2 */ A[k] = R[j]; j++;}
   }
   free(L);
   free(R);
}
```

```
void MergeSort(int* A, int p, int r){
   int q;
   if( p<r ){
      q = (p+r)/2;
      MergeSort(A, p, q);
      MergeSort(A, q+1, r);
      Merge(A, p, q, r);
   }
}

int Partition(int* A, int p, int r){
   int i, j, x;
   x = A[r];
   i = p-1;
   for(j=p; j<r; j++){
      if( A[j]<=x ){
         i++;
         swap(A, i, j);
      }
   }
   swap(A, i+1, r);
   return(i+1);
}

void QuickSort(int* A, int p, int r){
   int q;
   if( p<r ){
      q = Partition(A, p, r);
      QuickSort(A, p, q-1);
      QuickSort(A, q+1, r);
   }
}

void CountingSort(int* A, int* B, int n, int k){
   int i, j;
   int* C = calloc(k+1, sizeof(int));
   assert(C!=NULL);
   for(i=0; i<=k; i++) C[i] = 0;
   for(j=0; j<n; j++) C[A[j]] = C[A[j]]+1;
   for(i=1; i<=k; i++) C[i] = C[i] + C[i-1];
   for(j=n-1; j>=0; j--){
      B[C[A[j]]-1] = A[j];
      C[A[j]]--;
   }
   free(C);
}

int main(int argc, char** argv){
   int i, n=9;
   int A[] = {9,5,3,6,7,4,8,1,2};
   int B[9];

   for(i=0; i<n; i++) printf("%d ", A[i]);
   printf("\n");

   QuickSort(A, 0, 8);

   for(i=0; i<n; i++) printf("%d ", A[i]);
   printf("\n");


   return(EXIT_SUCCESS);
}
```

COUNTING SORT MANAGES TO BEAT THE $n \lg n$ LOWER BOUND BY NOT RELYING ON COMPARISON OF ARRAY ELEMENTS. IN FACT COUNTING SORT DOES NO ARRAY COMPARISONS!

INSTEAD IT ASSUMES THAT THE INPUT ARRAY A CONTAINS ONLY INTEGERS IN THE RANGE 0 TO k (i.e. k is AN INPUT TO THE ALGORITHM.)

COUNTING SORT UTILIZES A LOCAL ARRAY $C[0 \ldots k]$. THE FIRST LOOP INITIALIZES C TO ALL ZEROS, AND THE SECOND LOOP SETS $C[i]$ = # ELEMENTS IN A WHICH ARE EQUAL TO i. THE THIRD LOOP SETS $C[i]$ = # ELEMENTS IN A WHICH ARE LESS THAN OR EQUAL TO i

THUS $C[A[i]]-1$ is INDEX FOR $A[i]$ IN THE OUTPUT ARRAY IS. THE LAST LOOP PLACES $A[i]$ IN $B[C[A[i]]-1]$, THEN DECREMENTS $C[A[i]]$ SO AS TO MAKE ROOM FOR ANY REPEATED ELEMENTS.

COUNTING SORT IS ALSO <u>STABLE</u> IN THE SENSE THAT ANY REPEATED ELEMENTS IN THE INPUT ARRAY APPEAR IN THE SAME ORDER IN THE OUTPUT ARRAY.

COST = $\Theta(n+k)$, IF $k = O(n)$ THIS BEATS $n \lg n$ LOWER BOUND.