```
#include <iostream>                  // Quicksort alternativ
using namespace std;
void swap(int *a, int *b) {          // Vertauschen *a mit *b
   int h = *a;
   *a = *b;
   *b = h;
}

int partition(int z[], int l, int r) {
    int x = z[r],                    // Vergleich mit x=z[r]
        i = l-1,
        j = r;
    while (1) {
       while (z[++i] < x)
          ;
       while (z[--j] > x)
          ;
       if (i < j)
          swap(z+i, z+j);
       else {
          swap(z+i, z+r);
          return i;
       }
    }
}

// alternativ zur vorherigen Funktion partition
int partition(int z[], int l, int r) {
    int x = z[l],                    //Vergleich mit x=z[l]
        i = l,
        j = r+1;
    while (1) {
       while (z[++i] < x)
          ;
       while (z[--j] > x)
          ;
       if (i < j)
          swap(z+i, z+j);
       else {
          swap(z+j, z+l);
          return j;
       }
    }
}

void quick_sort(int z[], int l, int r) {
   if (l < r) {
      int pivot = partition(z, l, r);
      quick_sort(z, l, pivot-1);
      quick_sort(z, pivot+1, r);
   }
}
```

```
// Funktion zum rekursiven Einlesen von int-Zahlen ueber
// Tastatur, maximal ULONG_MAX == 4 294 967 295 (0xffffffff)
// Zahlen (falls der Stack-Speicher reicht !)

// f ist Parameter fuer Ein-/Ausgabe, beschreibt dynamisch
// vereinbarten Vektor mit n Elementen
//
// n ist Parameter fuer Ein-/Ausgabe, beschreibt die Anzahl
// der gelesenen Zahlen, dient beim Anlegen des Vektors
// als Grenze, wird auch ausserhalb von read() fuer das
// Durchlaufen des Vektors values[0..n-1] genutzt

void read(int * &f, unsigned long &n){ // 2 Referenzparameter
      int i = 0;               // lokaler Speicher fuer i
      unsigned long j = n;     // Index n nach j
      cout<<"int-Zahl = "; cin>>i; // Lesen der naechsten Zahl
      if(!cin.eof()){   // kein Dateiende gelesen
                              n++;   // n := n + 1
                              cin.clear(); cin.ignore(INT_MAX, '\n');
                              read(f, n);  // rekursiver Aufruf
      }
      if(cin.eof()){   // EOF (F6) gelesen
                  f = new int[n];// anlegen des int-Vektors f
                  cin.clear();   // cin.eof() zurueck auf false
                  if(f==0){      // Speicherplatzzuweisung fuer
                                    // Vektor f nicht moeglich !
                     cout<<"kein Speicherplatz bei n = "<<n<<endl;
                     n=0UL;      // n:=0, da kein Speicherplatz
                  }
                  return;
      }
      f[j] = i;  // in der Ebene j gelesener Wert i nach f[j]
}

// Kleiner Test: Lass es quicken!
void main() {
      unsigned long n = 0UL;
      int *values = 0;       // Zeiger auf Vektor values == 0
      read(values, n);       // Aufruf des rekursiven Lesens von Tastatur
      // Ausgabe der Werte (vorher)
      cout<<"Werte vor dem Sortieren:\n";
      for (unsigned long i = 0UL; i < n; i++)
        cout<<values[i]<<"  ";
      // Sortiere alle (count!) Elemente von values
      quick_sort(values, 0, n-1);
      // Ausgabe der Werte (nachher)
      cout<<"\n\nWerte nach dem Sortieren:\n";
      for (unsigned long i = 0UL; i < n; i++)
        cout<<values[i]<<"  ";
      delete [] values; values=0; // Freigabe des Vektors
      cin.get();
}
```

qsort in C / C++

Realisiert den Quicksort Algorithmus.

```
void qsort(                     /* Funktionsname, kein return-Wert          */
           void *base,          /* Adresse (0. El.) des zu sortierenden Arrays */
           size_t num,          /* Anzahl der Array-Elemente                */
           size_t width,        /* Größe eines Array-Elementes in Byte      */
           int (*compare )(     /* Zeiger auf Vergleichsfunktion, Typ int   */
             const void *elem1, /* 1.Arg., Zeiger auf zu sortierendes Element */
              const void *elem2 /* 2.Arg., Zeiger auf Vergleichsel. zum elem1 */
           )                    /* Ende der Parameterliste von compare      */
);                              /* Ende der Parameterliste von qsort        */
```

Routine	Required Header	Compatibility
qsort	<stdlib.h> oder <search.h>	ANSI, Windows 7

Bemerkungen:

The **qsort** function implements a quick-sort algorithm to sort an array of *num* elements, each of *width* bytes. The argument *base* is a pointer to the base of the array to be sorted.

qsort overwrites this array with the sorted elements. The argument *compare* is a pointer to a user-supplied routine that compares two array elements and returns a value specifying their relationship. **qsort** calls the *compare* routine one or more times during the sort, passing pointers to two array elements on each call:

*compare***((void *)** *elem1***, (void *)** *elem2* **);**

The routine must compare the elements, then return one of the following values:

Return Value	Description
< 0	*elem1 < elem2*
0	*elem1 == elem2*
> 0	*elem1 > elem2*

The array is sorted in increasing order, as defined by the comparison function. To sort an array in decreasing order, reverse the sense of “greater than„ and “less than„ in the comparison function.

Jeder Zeiger kann in den Typ **void *** und zurück umgewandelt werden, ohne daß Information verloren geht. Deshalb kann **qsort** aufgerufen und dabei die Argumente explizit in den Typ **void *** umgewandelt werden.

```c
#include <stdio.h>              // Beispiel qsort1
#include <stdlib.h>
#include <string.h>

int sort_str_up(const void *, const void *);
int sort_str_down(const char *, const char *);
int sort_int_up(const int *, const int *);
int sort_int_down(const void *, const void *);

char list[][4]={"cat","cha","cab","cap","can"};

int vektor[]={9, 8, -4, 5, 10, 7, 1, 12 };

void main() { int i;
  printf("Steigende Sortierung des char * - Vektors:\n");
  qsort((void *)list, sizeof(list)/sizeof(list[0]), sizeof(list[0]),
        &sort_str_up);
  for(i=0; i<sizeof(list)/sizeof(list[0]); i++) printf("%s ",list[i]);

  printf("\n\nFallende Sortierung des char * - Vektors:\n");
  qsort(list, sizeof(list)/sizeof(list[0]), sizeof(list[0]), sort_str_down);
  for(i=0; i<sizeof(list)/sizeof(list[0]); i++) printf("%s ",list[i]);

  printf("\n\nSteigende Sortierung des int - Vektors:\n");
  qsort(vektor, sizeof(vektor)/sizeof(int), sizeof(int), sort_int_up);
  for(i=0; i<sizeof(vektor)/sizeof(int); i++) printf("%d ", vektor[i]);

  printf("\n\nFallende Sortierung des int - Vektors:\n");
  qsort((void *)vektor, sizeof(vektor)/sizeof(int), sizeof(int),
        sort_int_down);
  for(i=0; i<sizeof(vektor)/sizeof(int); i++) printf("%d ", vektor[i]);
}

int sort_str_up(const void *a, const void *b)
{
      return( strcmp((char *)a, (char *)b));
}

int sort_str_down(const char *a, const char *b)
{
      return( strcmp(b, a));
}

int sort_int_up(const int *a, const int *b)
{
      return((*a<*b)?-1:((*a>*b)?1:0));
}

int sort_int_down(const void *b, const void *a)
{
      return((*(int *)a<*(int *)b)?-1:((*(int *)a>*(int *)b)?1:0));
}
```

```
/*
Steigende Sortierung des char * - Vektors:
cab can cap cat cha

Fallende Sortierung des char * - Vektors:
cha cat cap can cab

Steigende Sortierung des int - Vektors:
-4 1 5 7 8 9 10 12

Fallende Sortierung des int - Vektors:
12 10 9 8 7 5 1 -4
*/

-------------------------------------------------------------------------

// Beispiel qsort.cpp
#include<iostream>
using namespace std;

#include<cstdlib>   // enthält Prototyp von qsort()

// Definition der Vergleichsfunktion
int icmp(const void *a, const void *b)
{
    // Typumwandlung der Zeiger auf void in Zeiger auf int
    // und anschließende Dereferenzierung (von rechts lesen)
    int ia = *(int *)a;
    int ib = *(int *)b;

    // Vergleich und Ergebnisrückgabe ( > 0, = 0, oder < 0)
    if(ia == ib) return 0;

    return ia > ib ? 1 : -1;
}

void main()
{
    int ifeld[] = {100,22,3,44,6,9,2,1,8,9};

    // Die Feldgröße ist die Anzahl der Elemente des Feldes.
    // Feldgröße = sizeof(Feld) / sizeof(ein Element)

    int Groesse = sizeof(ifeld)/sizeof(ifeld[0]);

    // Aufruf von qsort():
    qsort(ifeld, Groesse, sizeof(ifeld[0]), icmp);

    // Ausgabe des sortierten Feldes:
    for (int i = 0; i < Groesse; ++i)
         cout << " " << ifeld[i];
    cout << endl;

    cin.ignore();
}

// 1 2 3 6 8 9 9 22 44 100
```