```
----------------------------------------------------------------
sort_algorithms_3.h: quicksort based on median-of-three pivot
----------------------------------------------------------------

#ifndef __SORT_3_H__
#define __SORT_3_H__

#include <vector>

template <typename T>
int partition(std::vector<T> &A, int left, int right) {
  // sort: order left, middle and right elements
  int middle = (left+right)/2;

  if (A[middle] < A[left])  swap(A[middle], A[left]);
  if (A[right] < A[left])   swap(A[right], A[left]);
  if (A[right] < A[middle]) swap(A[right], A[middle]);

  if (right-left+1 <= 3)
    return middle;

  // select pivot: median-of-three
  int pindex = middle;
  T pivot = A[pindex];

  // partition A: {<=}, {pivot}, {=>}
  swap(A[pindex], A[right-1]);

  int i = left;
  int j = right-1;

  while (1) {
    while (A[++i] < pivot) { }
    while (pivot < A[--j]) { }
    if (i>=j) break;
    swap(A[i], A[j]);
  }

  pindex = i;
  swap(A[pindex], A[right-1]);

  return pindex;
}

template <typename T>
void quicksort(std::vector<T> &A, int left, int right) {
  if (left < right) {
    int pindex = partition(A, left, right);
    quicksort(A, left, pindex-1);
    quicksort(A, pindex+1, right);
  }
}

template <typename T>
void quicksort(std::vector<T> &A) {
  quicksort(A, 0, A.size()-1);
}

#endif


----------------------------------------------------------------
Hint: Quicksort works by recursively selecting and placing a
sublist pivot in its proper place in the sorted list. Each time,
the remaining sublist data is partioned (reorganized) such that
{data <= pivot} is to the left of the pivot and {pivot <= data}
is to the right of the pivot.


Hint: Ideally, the pivot is the median but finding it requires
sorting. Instead, the above code uses the median of the left,
middle, and right data elements. You select the pivot randomly
in Lab 2. Both algorithms work well in practice.


Hint: Since A[left] <= pivot <= A[right] by design, these need
not be considered when partitioning. Also, the inner while loops
don’t need explicit bounds checks since a break condition will
be encountered when the left and right sublist ends are reached.
This may not true when the pivot is chosen differently. In fact,
you have to rethink this code for Lab 2.


Hint: The partition function can be merged in with the recursive
quicksort function. You will do this in Lab 2.
----------------------------------------------------------------
```

```
----------------------------------------------------------------
sort_usage.cpp: simple driver code for testing sort algorithms
----------------------------------------------------------------

#include <...>
using namespace std;

#include "sort_algorithms_1.h"
#include "sort_algorithms_3.h"

template <typename T>
void readdata(string &fname, vector<T> &A) { ... }

template <typename T>
void sortdata(vector<T> &A, string &algname) {
  if (algname.compare("insertion") == 0) {
    insertion(A);
  } else if (algname.compare("qsort") == 0) {
    quicksort(A);
  }
}

template <typename T>
void printdata(T p1, T p2, string &fname) { ... }

int main(int argc, char *argv[]) {
  if (argc != 3) {
    cerr << "usage: " << argv[0]
         << " -insertion|qsort file.txt\n";
    return 0;
  }

  string algname(&argv[1][1]);
  string fname_in(argv[2]);

  vector<string> A;

  readdata(fname_in, A);
  sortdata(A, algname);

  string fname_out = algname + "_" + fname_in;

  printdata(A.begin(), A.end(), fname_out);
}
```

```
----------------------------------------------------------------
select_algorithms.h: quickselect
----------------------------------------------------------------

#ifndef __SELECT_H__
#define __SELECT_H__

#include <vector>

#include "sort_algorithms_3.h"

template <typename T>
void quickselect(std::vector<T> &A, int k) {
  int left = 0, right = (int)A.size()-1;

  while (1) {
    int pindex = partition(A, left, right);

    if (pindex == k)
      return;

    if (k < pindex) right = pindex-1;
    else            left = pindex+1;
  }
}

#endif


----------------------------------------------------------------
Hint: Quicksort can be modified to produce a partially sorted
list for which the kth element is guaranteed to be in the right
place. Pick a pivot and partition the data. If the pivot is in
the kth place, stop and return. Otherwise continue with the left
or the right sublist. The result is known as quickselect.


Hint: As shown above, quickselect can be implemented iteratively
by updating the left and right indices.


Hint: In Lab 2, you will use quickselect to narrow the range of
data being sorted to k0:k1 by first partioning the data so that
data[0:k0-1] is less than or equal to data[k0:k1] which is less
than or equal to data[k1+1:N-1].
----------------------------------------------------------------
```

```
----------------------------------------------------------------
select_usage.cpp: driver code for testing quickselect
----------------------------------------------------------------

#include <...>
using namespace std;

#include "sort_algorithms_3.h"
#include "select_algorithms.h"

template <typename T>
void readdata(string &fname, vector<T> &A) { ... }

template <typename T>
void printdata(T p1, T p2, string &fname) { ... }

int main(int argc, char *argv[]) {
  if (argc != 4) {
    cerr << "usage: " << argv[0]
         << " -qsort|qselect kth file.txt\n";
    return 0;
  }

  string algname(&argv[1][1]);
  string fname_in(argv[3]);

  int kth = atoi(argv[2]);

  vector<string> A;

  readdata(fname_in, A);

  if (algname.compare("qsort") == 0)
    quicksort(A);
  else
  if (algname.compare("qselect") == 0)
    quickselect(A, kth-1);
  else
    return 0;

  cout << kth << ": " << A[kth-1] << "\n";

  string fname_out = algname + "_" + fname_in;
  printdata(A.begin(), A.end(), fname_out);
}
```

```
unix> ./select_usage -qsort 12 names.txt
12: AISHA

unix> ./select_usage -qselect 12 names.txt
12: AISHA


unix> paste qsort_names.txt qselect_names.txt |\
awk '{printf "%-15s %-15s\n", $1, $2}' | cat -n | head -20

        qsort           qselect

     1  ABE             ADAMS
     2  ABEL            ABEL
     3  ABRAHAM         ABE
     4  ADAM            ADDIE
     5  ADAMS           ADELINE
     6  ADDIE           ADAM
     7  ADELINE         ABRAHAM
     8  ADKINS          ADKINS
     9  AGNES           AGNES
    10  AHMED           AHMED
    11  AIDA            AIDA
    12  AISHA           AISHA
    13  AL              AL
    14  ALBA            ALBA
    15  ALBERT          ALBERT
    16  ALBERTO         ALEXANDER
    17  ALEXANDER       ALFONSO
    18  ALEXANDRA       ALEXANDRA
    19  ALFONSO         ALBERTO
    20  ALFONZO         ALFONZO

----------------------------------------------------------------
Hint: Notice how quickselect doesn't place all data in the right
place. However, the element of interest is where it should be.
----------------------------------------------------------------
```