```
#include "sortlib.h"

void Selection(int A[ ], int N) {
    for (int Last = N-1; Last >= 1; --Last) {
        int Largest = 0;
        for (int Current=1; Current<=Last; Current++) {
            if (A[Current] > A[Largest])
            Largest = Current;
        }
        int Temp = A[Last];
        A[Last] = A[Largest];
        A[Largest] = Temp;
    }
}

void Bubble(int A[ ], int N) {
    bool Sorted = false;
    for (int Pass = 1; (Pass < N) && (!Sorted); ++Pass) {
        Sorted = true;
        for (int Current=0; Current < N-Pass; ++Current) {
            if (A[Current] > A[Current+1]) {
                int Temp = A[Current];
                A[Current] = A[Current+1];
                A[Current+1] = Temp;
                Sorted = false;
            }
        }
    }
}

void Insertion(int A[ ], int N) {
    for (int Pick=1; Pick<N; ++Pick) {
        int Current = Pick;
        int Temp = A[Pick];
        for ( ; (Current > 0) && (A[Current-1]>Temp) ; --Current)
            A[Current] = A[Current-1];
        A[Current] = Temp;
    }
}

void Shell(int A[ ], int N) {
    int step = 4;
    while (step > 0) {
        for (int i = 0 ; i < step ; i++) {
            int j = 1;
            int Pick= j * step + i;
            for (; Pick < N ; ++j, Pick= j * step + i) {
                int Current = Pick;
                int Temp = A[Pick];
                for ( ; (Current > i) && (A[Current - step] > Temp) ; Current -=step)
                    A[Current] = A[Current-step];
                A[Current] = Temp;
            }
        }

        if (step/2 != 0) step = step/2;
        else if (step == 1) step = 0;
        else step = 1;
    }
}

int Temp[MAX_TEMP_DATA_SIZE_FOR_MERGE];

void Merge_Sub (int A[ ], int F, int Mid, int L) {
    int First1 = F; int Last1 = Mid;
    int First2 = Mid + 1; int Last2 = L;
    int Index = First1;
    for (; (First1 <= Last1) && (First2 <= Last2); ++Index) {
        if (A[First1] < A[First2]) {
```

```
            Temp[Index] = A[First1];
            ++First1;
        } else {
            Temp[Index] = A[First2];
            ++First2;
        }
    }
    for (; First1 <= Last1; ++First1, ++Index)
        Temp[Index] = A[First1];
    for (; First2 <= Last2; ++First2, ++Index)
        Temp[Index] = A[First2];
    for (Index = F; Index <= L; ++Index)
        A[Index] = Temp[Index];
}

void MergeR(int A[ ], int First, int Last) {
    if (First < Last) {
        int Middle = (First + Last) / 2;
        MergeR(A, First, Middle);
        MergeR(A, Middle+1, Last);
        Merge_Sub(A, First, Middle, Last);
    }
}

void Merge(int A[ ], int N) {
    MergeR(A, 0, N-1);
}

int Quick_Sub(int A[ ], int First, int Last) {
    int low, high, p;
    p = A[Last];
    low = First;
    high = Last-1;
    while (low < high) {
        while (A[low] < p) low++;
        while (p < A[high]) high--;
        if (low < high) {
            int Temp = A[low];
            A[low] = A[high];
            A[high] = Temp;
            low++;
            high--;
        }
    }
    int Temp = A[low];
    A[low] = A[Last];
    A[Last] = Temp;

    return low;
}

void QuickR(int A[ ], int First, int Last) {
    if (First < Last) {
        int PivotIndex= Quick_Sub(A, First, Last);
        QuickR(A, First, PivotIndex-1);
        QuickR(A, PivotIndex+1, Last);
    }
}

void Quick(int A[ ], int N) {
    QuickR(A, 0, N-1);
}
```