Partitioning
and Divide-and-Conquer Strategies

Partitioning

Partitioning simply divides the problem into parts.

Divide and Conquer

Characterized by dividing problem into sub-problems of same form as larger problem. Further divisions into still smaller sub-problems, usually done by recursion.

Recursive divide and conquer amenable to parallelization because separate processes can be used for divided parts. Also usually data is naturally localized.

Partitioning/Divide and Conquer Examples

Many possibilities.

- Operations on sequences of number such as simply adding them together

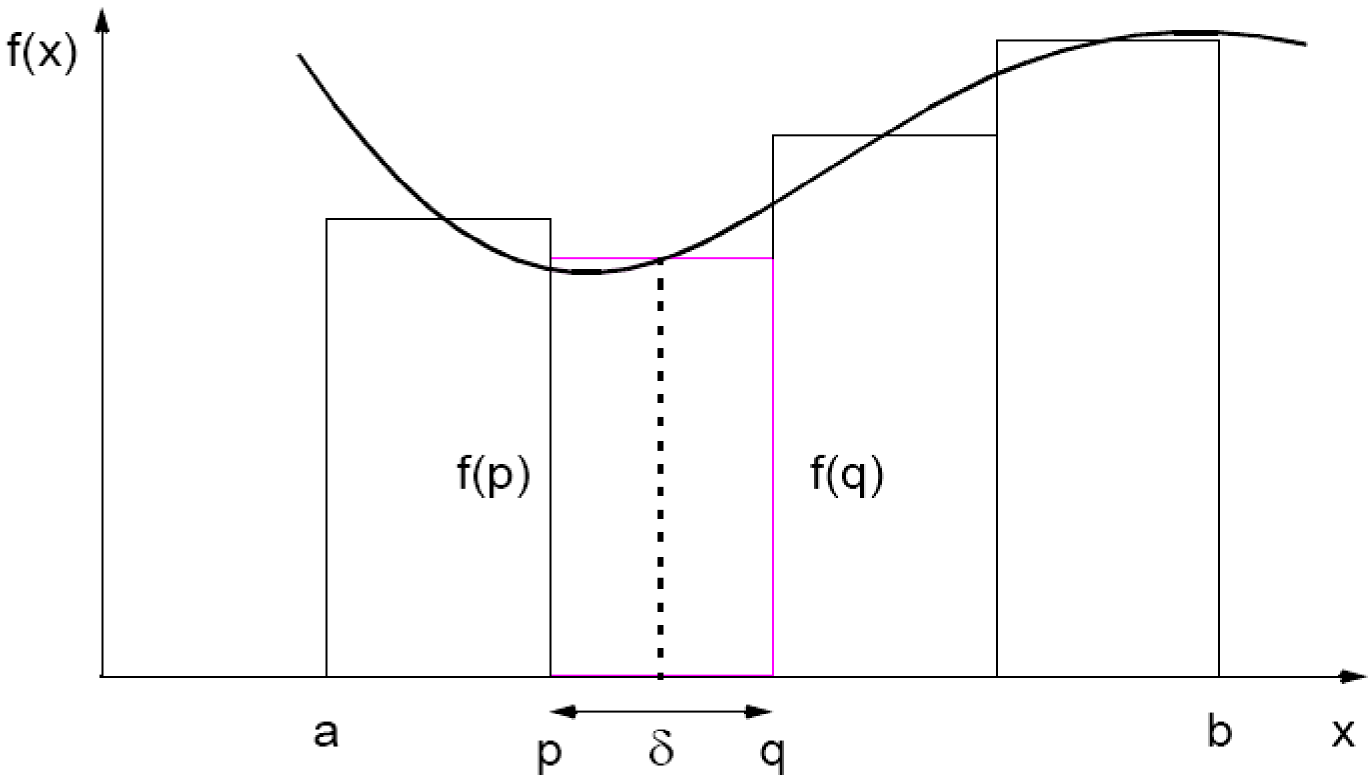
- Several sorting algorithms can often be partitioned or constructed in a recursive fashion

- Numerical integration

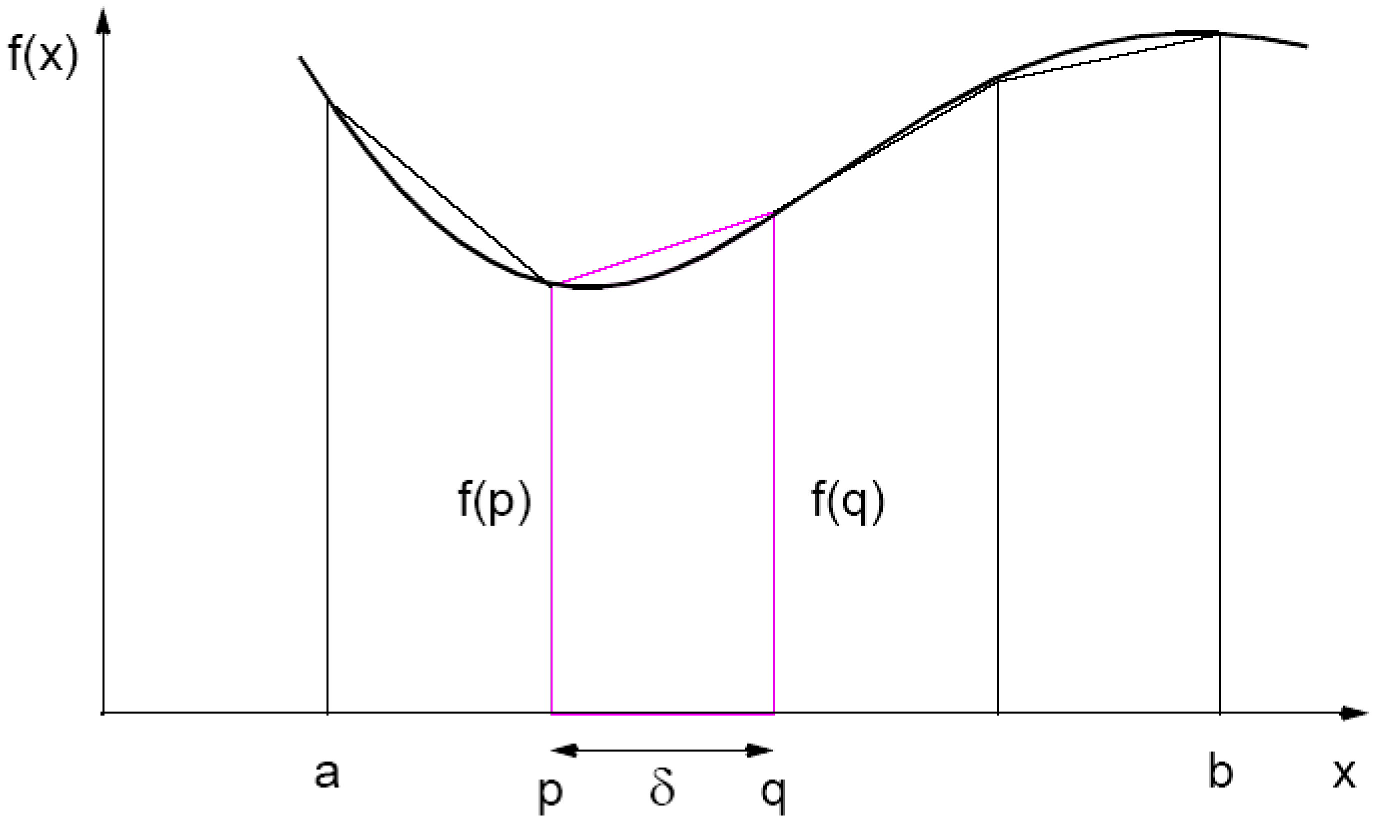
- *N*-body problem

Numerical integration using rectangles

Each region calculated using an approximation given by rectangles:
Aligning the rectangles:



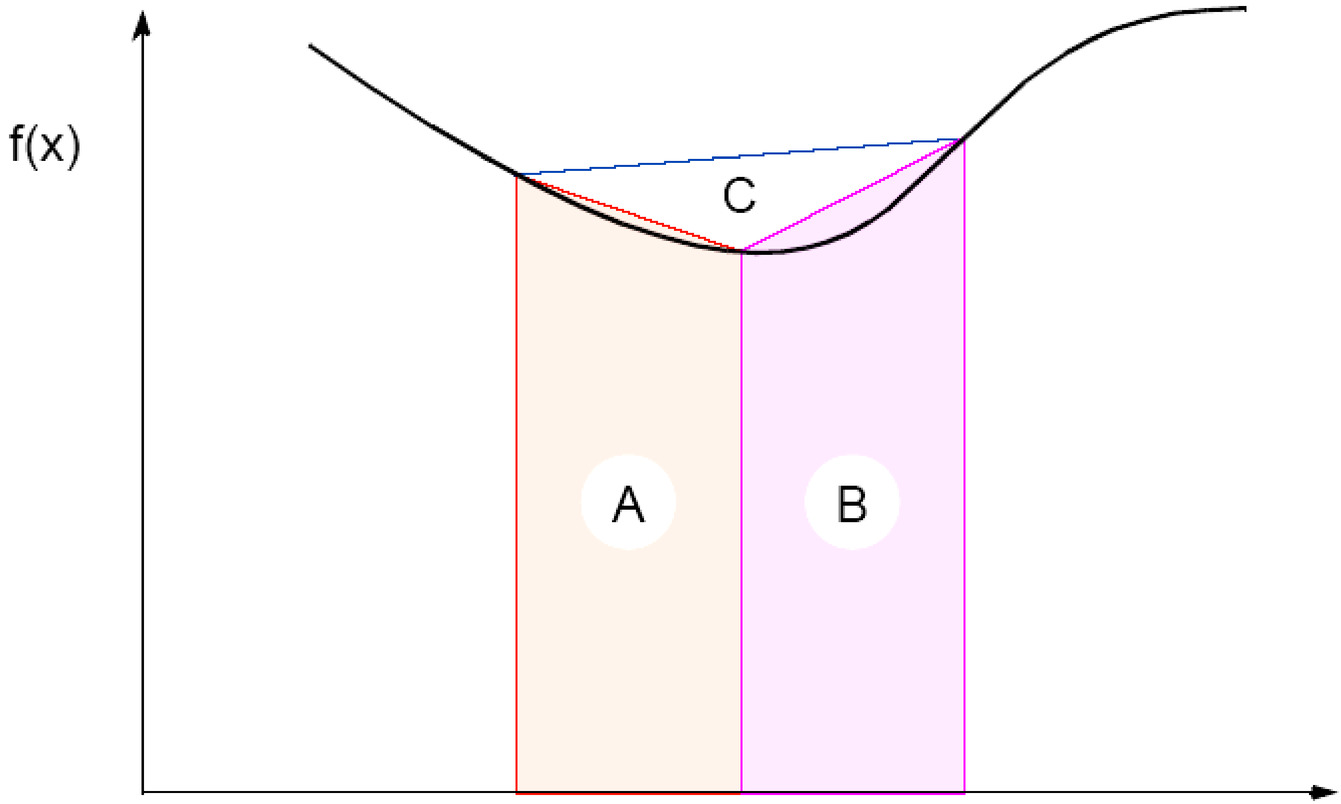
Numerical integration using trapezoidal method



May not be better!

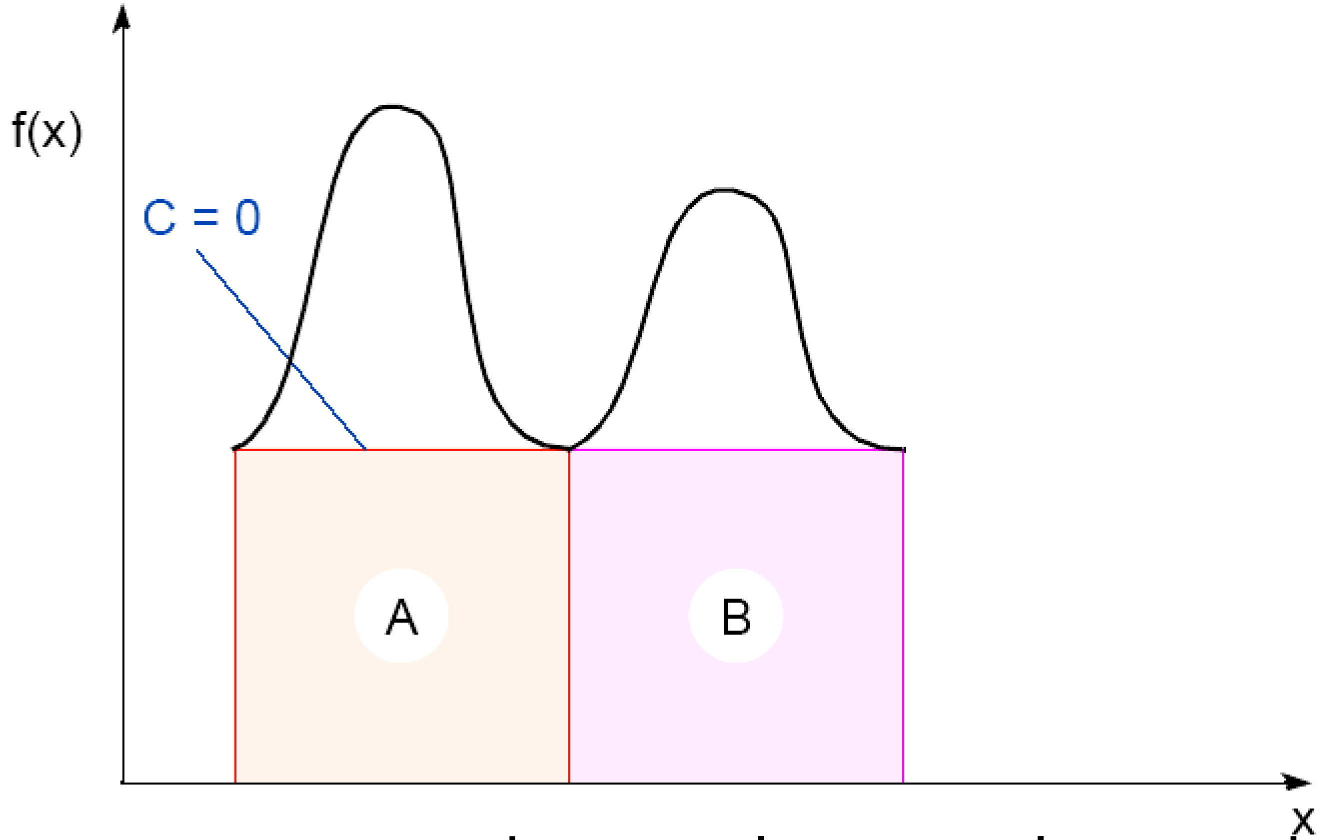
Adaptive Quadrature

Solution adapts to shape of curve. Use three areas, *A*, *B*, and *C*. Computation terminated when largest of *A* and *B* sufficiently close to sum of remain two areas .



Adaptive quadrature with false termination.

Some care might be needed in choosing when to terminate.



Might cause us to terminate early, as two large regions are the same (i.e., $C = 0$).

Simple program to compute π

Using C++ MPI routines


```
/*********************************************************************************
pi_calc.cpp calculates value of pi and compares with actual
value (to 25digits) of pi to give error. Integrates function f(x)=4/(1+x^2).
July 6, 2001 K. Spry CSCI3145
*********************************************************************************/
#include <math.h> //include files
#include <iostream.h>
#include "mpi.h“

void printit();                                   //function prototypes
int main(int argc, char *argv[])
{
double actual_pi = 3.141592653589793238462643;
                                                  //for comparison later
int n, rank, num_proc, i;
double temp_pi, calc_pi, int_size, part_sum, x;
char response = 'y', resp1 = 'y';
MPI::Init(argc, argv);                            //initiate MPI
```

```
num_proc = MPI::COMM_WORLD.Get_size();
rank = MPI::COMM_WORLD.Get_rank();
if (rank == 0) printit();           /* I am root node, print out welcome */

while (response == 'y') {
        if (resp1 == 'y') {
        if (rank == 0) {           /*I am root node*/
        cout <<"__________________________________________" <<endl;
        cout <<"\nEnter the number of intervals: (0 will exit)" << endl;
        cin >> n;}
} else n = 0;

MPI::COMM_WORLD.Bcast(&n, 1, MPI::INT, 0);     //broadcast n
if (n==0) break; //check for quit condition
```

```
else {
int_size = 1.0 / (double) n;                    //calcs interval size
part_sum = 0.0;

for (i = rank + 1; i <= n; i += num_proc)
 {                                              //calcs partial sums
        x = int_size * ((double)i - 0.5);
        part_sum += (4.0 / (1.0 + x*x));
}
temp_pi = int_size * part_sum;
                           //collects all partial sums computes pi

MPI::COMM_WORLD.Reduce(&temp_pi,&calc_pi, 1,
MPI::DOUBLE, MPI::SUM, 0);
```

```
if (rank == 0) {                                    /*I am server*/
cout << "pi is approximately " << calc_pi
<< ". Error is " << fabs(calc_pi - actual_pi)
<< endl
<<"_______________________________________"
<< endl;
}
}                                                   //end else
if (rank == 0) { /*I am root node*/
cout << "\nCompute with new intervals? (y/n)" << endl; cin >> resp1;
}
}//end while
MPI::Finalize();                                    //terminate MPI
return 0;
}                                                   //end main
```