Project 1 – Dynamic Integration

In this project, you are asked to implement functions that will enable you to apply numerical integration using randomly provided data.

The objective of this project is to learn about basics of numerical integration, Binary Search and AVL Trees, practice basic operations on these data structures, and compare and spot the performance differences.

Background Information

Numerical Integration

The basic idea of numerical integration is to approximate $\int_a^b f(x)dx$, $x \in [a,b]$, using the discrete values of $f(x)$ at finite x locations, i.e., $f(x_i)$, $x_i \in [a,b]$ and $i = 1,2,\ldots,N$ ($N < \infty$). There are several methods to calculate the numerical integration, for example, **trapezoidal rule** and **Simpson's rule** are the famous ones.

Numerical integration is especially useful;

1. When $f(x)$ is too complicated to integrate and to find an analytical expression its integral or when it is impossible to find an expression to it. In this case, $f(x)$ can be sampled and $\int_a^b f(x)dx$ can be calculated using a numerical integration method (also called rule), $\int_a^b f(x)dx \approx \sum_{i=1}^{N} a_i f(x_i)$. Here a_i, $i = 1,2,\ldots,N$, are the coefficients or weights of the numerical integration method. Usually, it is beneficial to choose the sample points equally spaced to reduce the numerical error.
2. When the values of $f(x)$ is known, instead of its expression, which is usually the case. In this case, only choice is using numerical integration.

In this project very primitive technique will be used to integrate the functions. First the function values after a sample point will be assumed to be constant and equal to function value at that point till the next sample point, i.e. assuming that x_i, $i = 1,2,\ldots,N$, are ordered, ($a < x_1 < x_2 < x_3 <\ldots< x_N < b$), then

$$f(x) = f(x_i), \text{ if } x_i \le x < x_{i+1}.$$

For the edge cases, it will be assumed that

$$f(x) = f(x_1), \text{ if } a \le x < x_1,$$

and

$$f(x) = f(x_n), \text{ if } x_N \le x \le b.$$

This will result in approximation to $f(x)$ using combination of rectangles. The integral of $f(x)$ in a given interval $[a,b]$ can be determined by calculating the area under covered by these rectangular shapes:

$$\int_a^b f(x)dx \approx (x_2 - a)f(x_1) + (x_3 - x_2)f(x_2) + (x_4 - x_3)f(x_3) + \cdots + (x_N - x_{N-1})f(x_{N-1}) + (b - x_N)f(x_N),$$

which can also be written as

$$\int_a^b f(x)dx \approx (x_2 - a)f(x_1) + \sum_{i=2}^{N-1}(x_{i+1} - x_i)f(x_i) + (b - x_N)f(x_N).$$

If there is a single sample of the function in the associated interval, then

$$\int_a^b f(x)dx \approx (b - a)f(x_1).$$

For example, let's assume $f(x) = x$, then

$$\int_a^b f(x)dx = \int_a^b xdx = \frac{x^2}{2}\bigg|_{x=a}^{b} = \frac{b^2-a^2}{2}.$$

Let's assume $a = 0$ and $b = 1$, $\int_a^b f(x)dx = \frac{1}{2}$. Using the procedure explained above:

- If there is only one sample at $x_1 = 0.435$ and $f(x_1) = x_1 = 0.435$, the approximation to function will be as shown in Figure 1. The result of the integral $\int_a^b f(x)dx \approx (b - a)f(x_1)$ is the area under the blue curve, which is 0.435.

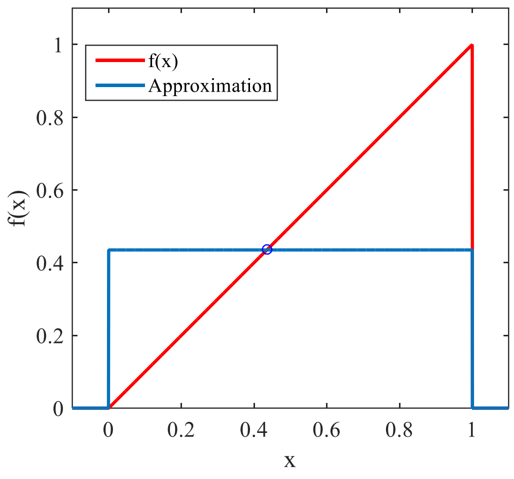

Figure 1. Approximation of $f(x)$ for $N = 1$.

- If there are two samples at $x_1 = 0.3$ and $x_2 = 0.79$, then $f(x_1) = 0.3$ and $f(x_2) = 0.79$. There are two rectangles that approximates the function as shown in Figure 2. The result of the integral $\int_a^b f(x)dx$ is the area under these two rectangles, which is 0.4029.

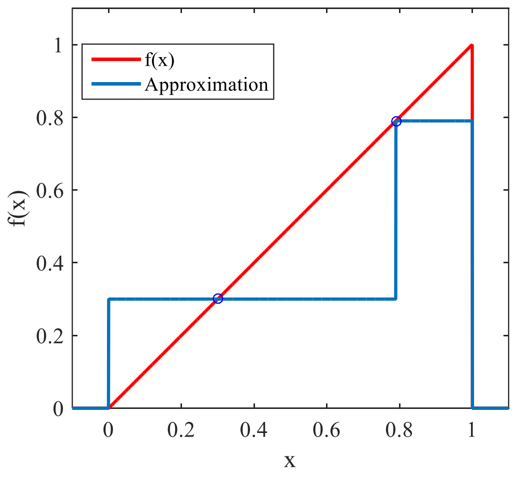

Figure 2. Approximation of $f(x)$ for $N = 2$.

- If $N = 3$, and $x_1 = 0.23,\ x_2 = 0.489,\ x_3 = 0.9$, then $f(x_1) = 0.23$, $f(x_2) = 0.489$, and $f(x_3) = 0.9$. There are three rectangles that approximates the function as shown in Figure 3. The result of the integral $\int_a^b f(x)dx$ is the area under these rectangles, which is 0.4034. Note that the first rectangle starts from $a = 0$.

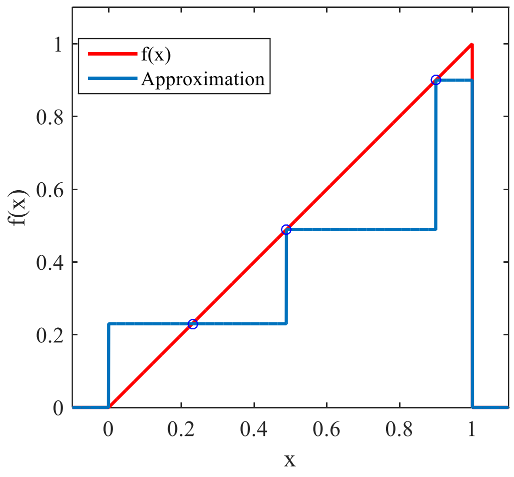

Figure 3. Approximation of $f(x)$ for $N = 3$.

- If there are enough number of samples than the integral will converge to its actual value. For $N = 20$, as shown in Figure 4, the result of the integral $\int_a^b f(x)dx$ is 0.4775.

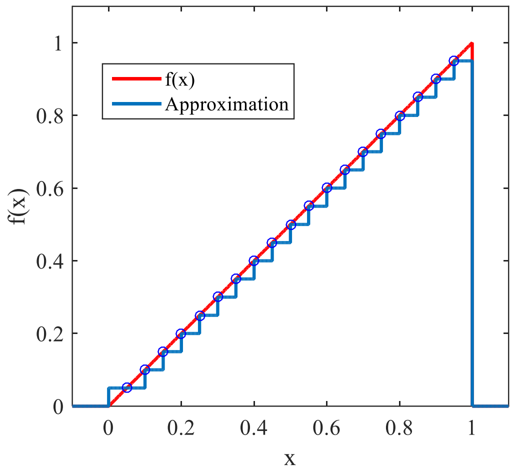

Figure 4. Approximation of $f(x)$ for $N = 20$.

!Warnings!

- In the procedure given above, it is strictly assumed that the samples are ordered (sorted) with respect to x.
- If the samples are not ordered, either they should be sorted or operations should be updated such that the contribution of old sample is compensated. This can be done by dynamically updating the integral value as the new sample point taken into account.

Dynamic Integration Update Procedure

Let's assume that the expression of $f(x)$ is **not known**, and x_i, $i = 1,2,\ldots,N$, are **ordered**, ($a \le x_1 < x_2 < x_3 <\ldots< x_N \le b$), the values of $f(x_i)$ are **known** and the integral of $f(x)$ is calculated using the given procedure and is equal to $IntVal$. Then there is a new value of the function $f(x_{new})$ at x_{new} appears. The integral value should be updated to include the effect of the new sample. The procedure for this update can be given as:

1. Determine the location of the new sample: This requires a search in x_i, $i = 1,2,\ldots,N$, set.
2. Once the location is found ($x_k < x_{new} < x_{k+1}$) insert the new value and update $N \leftarrow N + 1$. For this position, new location of x_{new} will be $k + 1$ ($x_k < x_{new} < x_{k+1}$ should be updated as ($x_k < x_{new} = x_{k+1} < x_{k+2}$). To not confuse with indices let's update them as $x_{nsmaller} < x_{new} < x_{nlarger}$, where nsmaller stands for next smaller and nlarger stands for next larger.
3. Because of $x_{nsmaller}$ there is a contribution to the integral value in the interval $[x_{new}, x_{nlarger}]$ with the weight of $f(x_{nsmaller})$. This contribution should be subtracted from $IntVal$ as
$$IntVal = IntVal - \left(x_{nlarger} - x_{new}\right)f(x_{nsmaller})$$
Then the contribution of the new sample should be added as
$$IntVal = IntVal + \left(x_{nlarger} - x_{new}\right)f(x_{new}).$$
Summing these contributions will result in an update operation as
$$IntVal = IntVal + \left(x_{nlarger} - x_{new}\right)[f(x_{new}) - f(x_{nsmaller})].$$
Note that this update assumes $x_{nsmaller} < x_{new} < x_{nlarger}$, and $x_{nsmaller}$ and $x_{nlarger}$ exist.
4. Considering the numerical integral calculation rule, exceptional cases should be taken into account to calculate the correct contribution:
    a. If $x_{nlarger}$ doesn't exist and $x_{nsmaller}$ exists, then x_{new} is the last and largest element in the order, i.e. $x_{nsmaller} < x_{new} \leq b$.
    b. If $x_{nsmaller}$ doesn't exist and $x_{nlarger}$ exists, then x_{new} is the first and smallest element in the order, i.e. $a \leq x_{new} < x_{nlarger}$.
    c. If both $x_{nsmaller}$ and $x_{nlarger}$ do not exist, then x_{new} is only element in the set.

   Handling of these exceptional cases should conform with the procedure for integration.

Let's consider the example given for $N = 3$ with Figure 3. There are three samples $N = 3,$ and $x_1 = 0.23,\ x_2 = 0.489,\ x_3 = 0.9,$ then $f(x_1) = 0.23$, $f(x_2) = 0.489$, and $f(x_3) = 0.9$. This leads to an integral value 0.4034. $x_{new} = 0.65$ with the function value $f(x_{new}) = 0.65$. First, the location of x_{new} in the set can be found as $x_2 = 0.489 < x_{new} = 0.65 < x_3 = 0.8$. Then the integral value can be updated as
$$IntVal = 0.4034 + (0.9 - 0.65)[0.65 - 0.489] = 0.44365\ .$$

Project Design Rules:

- You will be working on implementing functions that will allow you to calculate the numerical integral or update the integral value in a given interval using **unordered** (**unsorted**) measurement data provided to you.

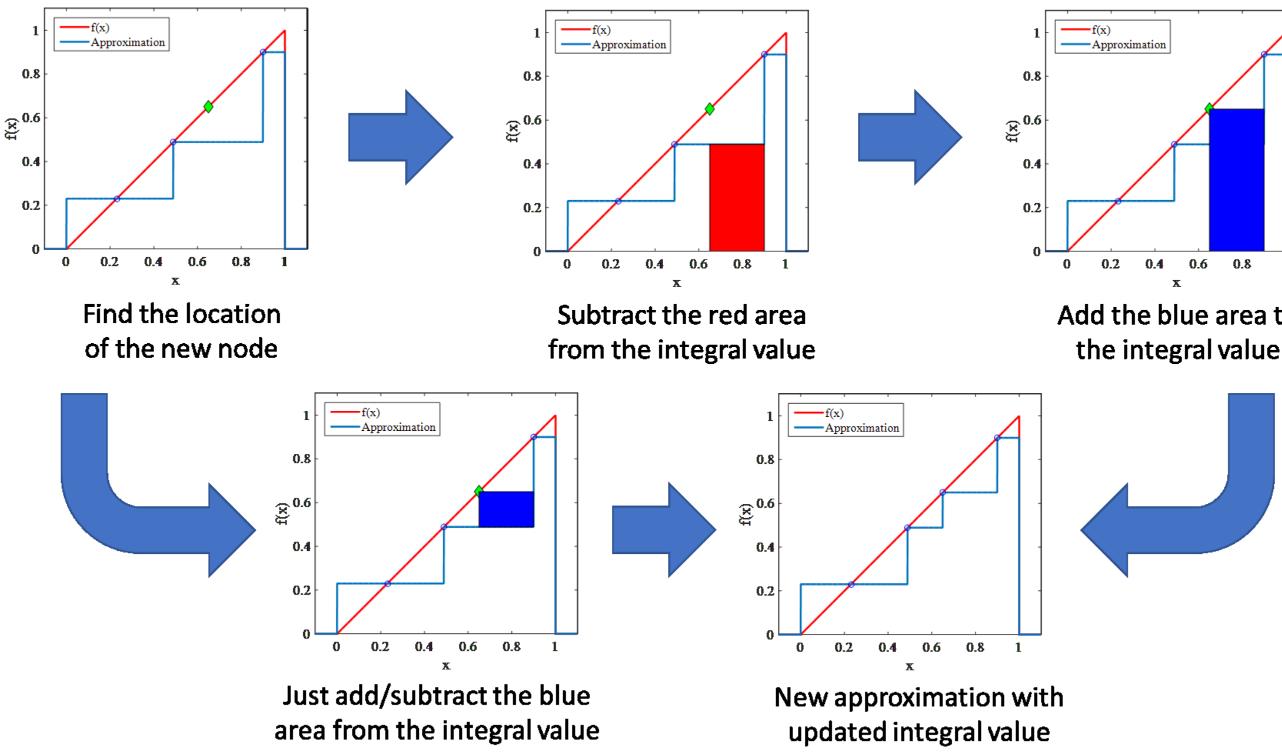
f(x)
Approximation
f(x)
x
Find the location of the new node
Subtract the red area from the integral value
Add the blue area to the integral value
Just add/subtract the blue area from the integral value
New approximation with updated integral value

Figure 5. Illustration of the procedure for dynamically updating the integral value.

- ○ The interested integral is given as

$$IntVal = \int_{\bar{t}_{lo}}^{\bar{t}_{up}} GTU(\bar{t})d\bar{t} = \frac{1}{3600}\int_{t_{lo}}^{t_{up}} GTU(t)dt$$

  where $\bar{t}$ is in hours and t is in seconds. Note that the time variable in the measurement data is given in seconds, therefore it is important to use the second integral expression given in seconds.

- Information about the measurement data:
  - ○ The data will be provided with a .txt file and every line denotes the time and value of the measurement,
    `'hhmmss    value'`
    Here, `'hhmmss'` denotes the time where `'hh'` denotes hour, `'mm'` denotes minute, and `'ss'` denotes second of the measurement data. For example,
    `'073528    0.02342349887574498'`
    denotes the time of measurement as 7 hours 35 minutes and 28 seconds (07:35:28), `0.02342349887574498` is the measurement value in “gt unites” (GTU). Time value is in 24 hours and given in 24-hour time format.
  - ○ The measurement period starts with a base time (`tbase`) and limited with a last time (`tceil`). These values should be parameters but default values are 07:00:00 for starting time and 22:00:00 for last time. All provided measurement values guaranteed to be between these values.
  - ○ You have to convert time values to distance to the base time in seconds. For example, of the measurement data is given at `'073528'` and base time is `'070000'`, then the key for this measurement value should be 2128 seconds.
  - ○ It is also guaranteed that for a given time there will be only one measurement value and a value for a given time won’t repeat in the provided data.

 - ○ The integral should be calculated between given upper and lower limits. The default values for these limits are `tceil` and `tbase`, respectively.

- You will be graded for each of the given functions so make sure to implement each of the functions **with exact names, parameters, and default values** -if given- in your python code.
- You should base your functions and operations on Binary Search Trees (BST) and AVL Trees using the provided files.
    - ○ BST : bstgtu.py
    - ○ AVL : avlgtu.py

  The modules with same file name will be used to grade and test your functions. Do not change the name of the files and be sure your naming is correct. **These files shouldn't be submitted as separate files or shouldn't be included into your submitted file.** If you copy these files into your submission or use any other tree modules, your submission will not be graded.
- You can use provided bstgtu and avlgtu modules and time module to measure timing, if asked. Specifically measure the CPU time using `perf_counter_ns()`. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- No additional modules should be used except time, bstgtu, avlgtu.
- Any other sorting or integrating, that does not based on BST and AVL trees and provided files, will not be graded.

BST and AVL Modules:

bstgtu,py and avlgtu.py files are based on bstsize_r.py and avl.py, which can be downloaded from the MIT website, respectively. You can find detailed information in the MIT website. There are small changes but related to the project, you should assign the measurement value to can use `BSTnode.gtu`, which is already defined in bstgtu,py. To set and get the value of `BSTnode.gtu`, you can use `BSTnode.set_gtu(gtuval)` and `BSTnode.get_gtu(self)` functions.

Functions:

Function Name	**Explanation**
`reltime(t,tbase='070000')`	This function will take two string parameters. It returns the **relative time** of `t` with respect to `tbase` in seconds as an integer. Both `t` and `tbase` are in 24 hour format. Inputs: ● `t` : string (`t>tbase`) ● `tbase` : string (default = `'070000'`) Return the relative time (`t-tbase`) as integer in seconds.

`read2tree(filename, TreeType, tbase='070000', tceil='220000')`	This function will take maximum four parameters, then • reads the data in the given `filename`, • creates a specified typed empty tree, • converts time to relative time with respect to `tbase` and insert it to the tree as a key of a node, • sets the measurement value of the key to the `gtu` parameter, • returns the tree structure. Inputs: ● `filename` : string ● `TreeType` : string ∈ `{'BST','AVL'}` ● `tbase` : string (default = `'070000'`) ● `tceil` : string (default = `'220000'`) Return the tree as specified tree object: ● `output : 'BST' or 'AVL'` object
`maximum(node)`	This function will take one parameter, which is a node object. Returns the node with the largest key in the sub-tree rooted by this node. Input: ● `node : 'BSTnode'` object Return `node` as specified node object.
`predecessor(node)`	This function will take one parameter, which is a node object. Returns the node with the largest key smaller than this node's key, or None if this has the smallest key in the tree. Input: ● `node : 'BSTnode'` object Return `node` as specified node object.
`dyna_int_updater(IntValue,tdo,tup, innode,nxsmaller,nxlarger)`	This function will take 6 parameters. Returns the updated integral value. Inputs are • `IntValue` : integral value, • `tdo` : lower limit of the integral, • `tup` : upper limit of the integral, • `innode` : node of the inserted new data, • `nxsmaller` : node that has the largest key smaller to the key of the new node,

	• `nxlarger` : node that has the smallest key larger from the key of the new node. Inputs: ● `IntValue` : float ● `tdo` : integer (`tdo≤tup`) • `tup` : integer (`tdo≤tup`) • `innode` : `'BSTnode'` object • `nxsmaller` : `'BSTnode'` object • `nxlarger` : `'BSTnode'` object Return the updated integral value as float.
`GTU_tree_rad_calc(MTree, tlo=None, thi=None, tbase="070000", tceil="220000")`	This function will take maximum 5 parameters Calculates the integral in the specified limits (`tlo`, `thi`) using the measurement values given with the tree (`MTree`). Returns the integral value, list of the updated integral values, and calculation time. Inputs are • `MTree` : tree object that holds the data, • `tlo` : lower limit of the integral (`tdo≥tbase`) (if not specified then `tdo=tbase`) • `thi` : upper limit of the integral (`tbase≤tdo<thi≤tceil`), (if not specified then `thi=tceil`) • `tbase` : the base time of the data, • `tceil` : upper limit of the data. Inputs: • `MTree` : `'BST' or 'AVL'` object • `tlo` : string (default = `None`) • `thi` : string (default = `None`) ● `tbase` : string (default = `'070000'`) ● `tceil` : string (default = `'220000'`) Return ● `output1` : integral value calculated using all data in `MTree`, ● `output2` : list of the updated integral values that is calculated as key values taken into account from `MTree`. `len(output2)` should be the number data that is taken into account to the calculate integral. The last element of this list should be equal to `output1`, ● `output3` : execution time in nanoseconds. Return ● `output1` : float, ● `output2` : list with float elements ● `output3` : integer

`GTU_tree_dyna_add(filename, MTree, IntValue=0, tlo=None, thi=None, tbase="070000", tceil="220000")`	This function will take maximum 7 parameters, then • reads the data in the given `filename`, • converts time to relative time with respect to `tbase` and insert it to the tree (`MTree`) as a key of a node, • sets the measurement value of the key to the `gtu` parameter, • updates the integral value in the specified limits (`tlo, thi`) • returns the integral value, list of the updated integral values, and calculation time. Inputs ● `filename` : file name for the new data, • `MTree` : tree object that holds the old data and will be updated with the data in `filename`, • `IntValue` : Integral value that is calculated with the initial entries of `MTree`, and will be updated with the data in `filename`, • `tlo` : lower limit of the integral (`tdo≥tbase`) (if not specified then `tdo=tbase`) • `thi` : upper limit of the integral (`tbase≤tdo<thi≤tceil`), (if not specified then `thi=tceil`) • `tbase` : the base time of the data, • `tceil` : upper limit of the data. Inputs: ● `filename` : string • `MTree` : `'BST' or 'AVL'` object • `IntValue` : float (default = `0`) • `tlo` : string (default = `None`) • `thi` : string (default = `None`) ● `tbase` : string (default = `'070000'`) ● `tceil` : string (default = `'220000'`) Return ● `output1` : updated integral value calculated using all data in `filename`, ● `output2` : list of the updated integral values that is calculated as key values taken into account. `len(output2)` should be the number of data in `filename` that is taken into account to update the integral value. The last element of this list should be equal to `output1`, ● `output3` : execution time in nanoseconds. Return ● `output1` : float, ● `output2` : list with float elements ● `output3` : integer

`GTU_int_calc(filename, TreeType, tlo=None, thi=None, tbase="070000", tceil="220000")`	This function will take maximum 6 parameters Calculates the integral in the specified limits (`tlo, thi`) using the measurement values using the data provided in `filename`. Returns the integral value, list of the updated integral values, and calculation time. Inputs are ● `filename` : file name for the data, ● `TreeType` : specifies the type of the tree ● `tlo` : lower limit of the integral (`tdo≥tbase`) (if not specified then `tdo=tbase`) ● `thi` : upper limit of the integral (`tbase≤tdo<thi≤tceil`), (if not specified then `thi=tceil`) ● `tbase` : the base time of the data, ● `tceil` : upper limit of the data. Inputs: ● `filename` : string ● `TreeType` : string ∈ `{'BST','AVL'}` ● `tlo` : string (default = `None`) ● `thi` : string (default = `None`) ● `tbase` : string (default = `'070000'`) ● `tceil` : string (default = `'220000'`) Return ● `output1` : integral value calculated using all data in `MTree`, ● `output2` : list of the updated integral values that is calculated as key values taken into account from `MTree`. `len(output2)` should be the number data that is taken into account to calculate integral. The last element of this list should be equal to `output1`, ● `output3` : execution time in nanoseconds. Return ● `output1` : float, ● `output2` : list with float elements ● `output3` : integer

```
>>> Mtree = read2tree('test_set0.txt','BST')
>>> print(Mtree)
         26406.
        /      \
      12420   48600
      /   \   /   \
    6480    35100
    /  \    /   \
  5400
  /  \
2700
/  \

>>> output1,output2,output3 = GTU_tree_rad_calc(Mtree)
>>> print(output1)
6.164985
>>> print(output2)
[0.075, 0.105, 0.303, 1.19655, 2.377485, 4.814985, 6.164985]
>>> print(output3)
25800

>>> output1,output2,output3 = GTU_int_calc('test_set0.txt','BST')
>>> print(output1)
6.164985
>>> print(output2)
[7.335, 5.435235, 6.051735, 6.655485, 6.2759849999999995, 6.239985, 6.164985]
>>> print(output3)
383100

>>> Mtree = read2tree('test_set0.txt','AVL')
>>> print(Mtree)
       ..26406..
      /         \
    6480       48600
    /   \      /   \
  5400 12420 35100
  /  \ /   \ /   \
2700
/  \
>>> output1,output2,output3 = GTU_tree_rad_calc(Mtree)
>>> print(output1)
6.164985
>>> print(output2)
[0.075, 0.105, 0.303, 1.19655, 2.377485, 4.814985, 6.164985]
>>> print(output3)
33700
```

```
>>> output1,output2,output3 = GTU_int_calc('test_set0.txt','AVL')
>>> print(output1)
6.164985
>>> print(output2)
[7.335, 5.435235, 6.051735, 6.655485, 6.2759849999999995, 6.239985, 6.164985]
>>> print(output3)
415900

>>> Mtree = read2tree('test_set0a.txt','BST')
>>> print(Mtree)
   26406.
  /      \
12420   48600
/   \   /   \
      35100
      /   \

>>> output1,output2,output3 = GTU_tree_rad_calc(Mtree)
>>> print(output1)
6.6554850000000005
>>> print(output2)
[1.68705, 2.867985, 5.305485, 6.6554850000000005]
>>> print(output3)
24700

>>> output1,output2,output3 = GTU_tree_dyna_add('test_set0b.txt',Mtree,output1)
>>> print(output1)
6.164985000000001
>>> print(output2)
[6.275985, 6.239985000000001, 6.164985000000001]
>>> print(output3)
444800
>>> print(Mtree)
         26406.
        /      \
      12420   48600
      /   \   /   \
    6480    35100
    /  \    /   \
  5400
  /  \
2700
/  \

>>> Mtree = read2tree('test_set0a.txt','AVL')
>>> print(Mtree)
   26406.
  /      \
12420   48600
/   \   /   \
      35100
      /   \

>>> output1,output2,output3 = GTU_tree_rad_calc(Mtree)
>>> print(output1)
6.6554850000000005
>>> print(output2)
[1.68705, 2.867985, 5.305485, 6.6554850000000005]
>>> print(output3)
24700

>>> output1,output2,output3 = GTU_tree_dyna_add('test_set0b.txt',Mtree,output1)
>>> print(output1)
6.164985000000001
>>> print(output2)
[6.275985, 6.239985000000001, 6.164985000000001]
>>> print(output3)
362200
>>> print(Mtree)
       ..26406..
      /         \
    6480       48600
    /   \      /   \
  5400 12420 35100
  /  \ /   \ /   \
2700
/  \

>>> node=Mtree.find(12420)
>>> print(node)
<BST Node, key:12420>
>>> print(node.get_gtu())
0.23
>>> print(maximum(node))
<BST Node, key:12420>
>>> print(node.successor())
<BST Node, key:26406>
>>> print(predecessor(node))
<BST Node, key:6480>
```