Quality Management Plan

Project Management Improvement Project

By: Matthew Fernandez, Matthew Ljuljic, Yehua Zhang, Nathan Gillette

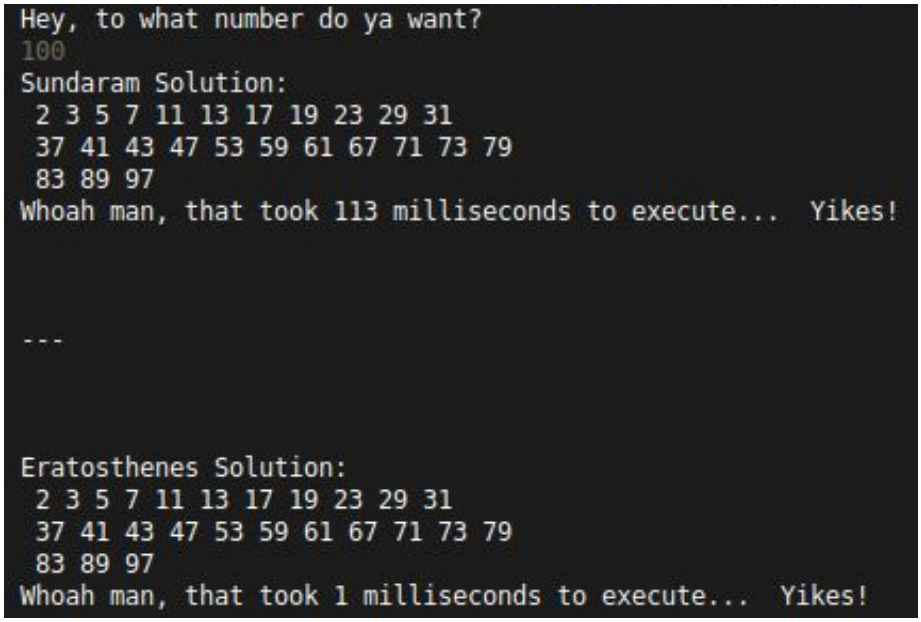
156.

157. 0.Table Of Contents

158.

1. Introduction

We are analyzing a prime number generator. The program asks the user how many prime numbers they want to generate. The user then enters a positive integer value, and the program will generate that amount of prime numbers.



One of the most important methods of this program is the genEratosthenes method. It is a method that takes an integer input and returns a BitSet which contains the set of all prime numbers that are within the range of the integer input. For example, if the user input 100, this method would generate all of the prime numbers between 2 and 100. The method iterates through each number in the range, and it would eliminate each of its multiples. All of the numbers not eliminated will be added to the BitSet and returned.

Eratosthenes set:

“1” is omitted from this set

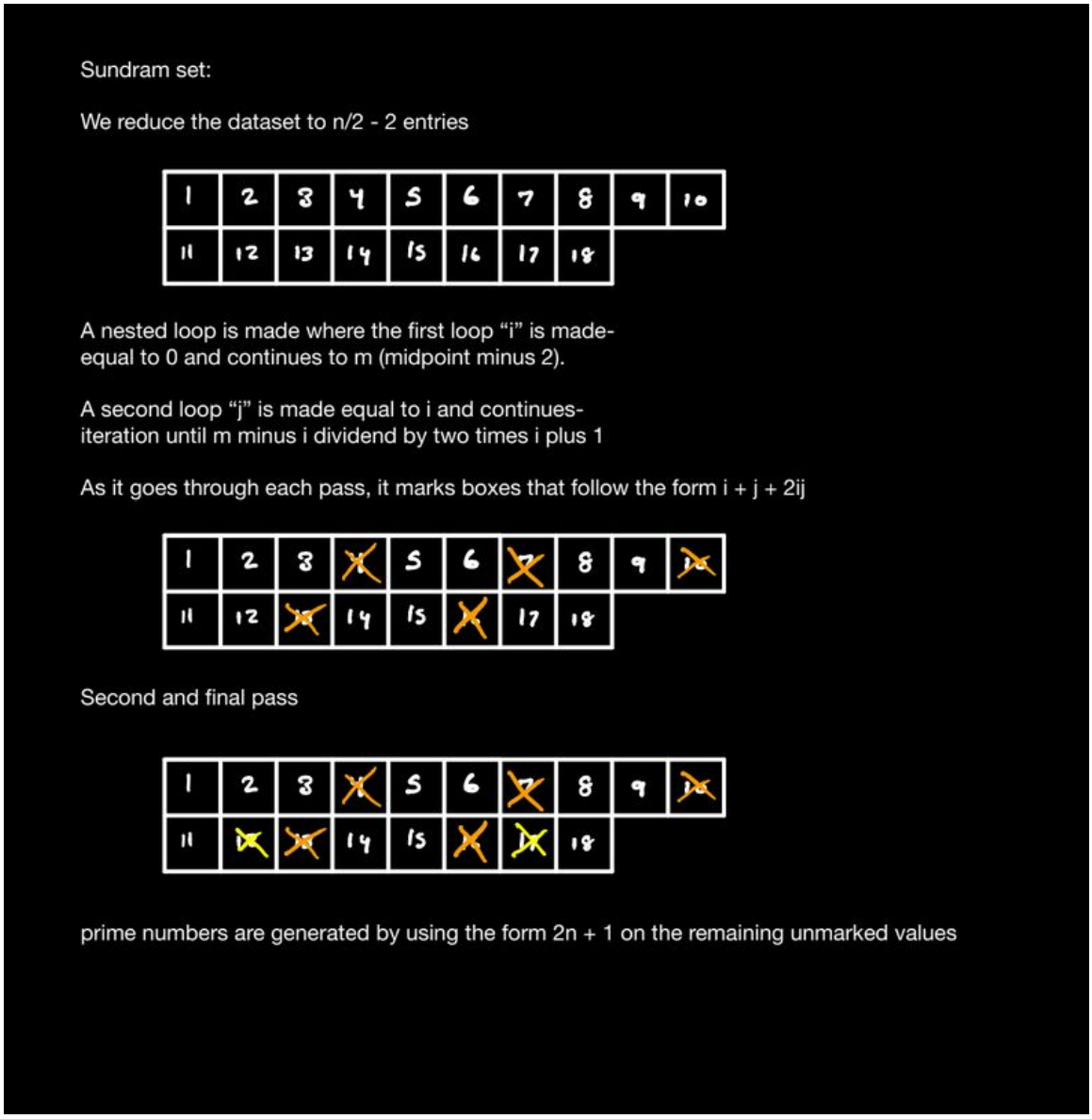
“2” is first unmarked number, all multiples are marked

“3” is next unmarked number, multiples are marked

“5” is selected, multiples are marked, algorithm stops at the square of the end number “40”

All remaining numbers are prime.

Another important method is the genSundaram method. This method takes an integer, computes all prime numbers within the range of 2 and that integer, and then returns the prime numbers in a BitSet. This method is identical in function to the genEratosthenes method, however the manner in which this method computes the prime numbers is different. This method takes the function i+j+2ij and uses that to generate all non-prime numbers within the specified range and eliminates them. The method then collects all of the remaining numbers and returns them as a BitSet because they are the set of prime numbers within the range.



Another method that is important to this program is the fileConstructorSundaram. This method takes a number from the user that represents the range of prime numbers the user wants to generate. The method then passes this to the genSundaram method and allows it to generate the BitSet of prime numbers. Once genSundaram is done, it returns the BitSet of prime numbers to this method. This method then writes the contents of the BitSet to a text file as well as displaying the contents in the terminal. This method is of void type so it returns nothing.

The final method that is important to this program is the fileConstructorEratosthenes. This method takes a number from the user that represents the range of prime numbers the user wants to generate. The method then passes this to the genEratosthenes method and allows it to generate the BitSet of prime numbers. Once genEratosthenes is done, it returns the BitSet of prime numbers to this method. This method then writes the contents of the BitSet to a text file as well as displaying the contents in the terminal. This method is of void type so it returns nothing.

159.

160.

161.

2. Quality Management Approach

162. 2.1 The identification of the system

The prime number generator has the components of two, one being Sieve of Eratosthenes and the another one Sieve of Sundrum, these are two different compute mode for generating prime numbers; One class named Alg; and 5 methods which are: genEratosthenes(), genSundaram(), fileConstructorSundaram(), fileConstructorEratosthenes(), main(). The program has one attribute which is an integer number, and operations are recursion, mathematical computation, nested loops, etc.

163. 2.2The synopsis of the system

The purpose of the prime number generator is to generate all the prime numbers in a given interval. The quality goals of this program is to generate prime numbers without errors as well as doesn't exceed compute ability of the target operating system and machine.

164. 2.3 Glossary

Sieve - something that sifts through data and filters it by some sort of metric

Sieve of Eratosthenes - A sieve that filters numbers based on if they are prime. This is done by iterating through the list of numbers, checking to see if a current number is marked, and marking all multiples of that number. Any unmarked numbers that are found are prime.

Sieve of Sundrum - A sieve that filters numbers based on if they are prime. This is done by having two separate counters that go through the list at the same time where the second counter cannot backtrack. A number is marked through each iteration using the form: i + j +2ij. Once the algorithm is done all unmarked numbers are adjusted by using the formula: 2i + 1. All of these calculated numbers are prime.

Bitset - an array where every position is only one bit in size (i.e. 0 or 1)

165. 2.4 Partners

Name	Contacts	Responsibilities
Nathan Gillette	ngillet2@oswego.edu	Introduction and Specification Based Testing
Matthew Fernandez	mfernan4@oswego.edu	Supplied code and Specification Based Testing
Matthew Ljuljic	mljuljic@oswego.edu	Testing Plan and Specification

		Based Testing
Yehua Zhang	yzhang9@oswego.edu	identification and synopsis

166. 2.5 Keys for Usability(numbered by importance)

Quality Property	Definition	Metric	Criterion
1. correctness	Executed test cases yield the same result that anticipated during their derivation.	Systematic derivation and execution of test cases for specification based, control flow based, and data flow based tests.	The component is considered correct, iff at least • 100% of test cases with high criticality pass • 80% of test cases with medium criticality pass • 40% of test cases with low criticality pass.
4. User friendliness	The program completely places control in users hands while guiding them on what to do.	Step-by-Step process of running the program and checking at each input junction if the user understand what is happening and what is asked of them	The system is considered user friendly if: The user can operate the system with full understanding of how their input is used and the subsequential output.
3. Reliability	The program's ability to be called multiple times without error as well as how well it handles errors while running.	At every stage of the program where the user has control inputting the wrong value and evaluating how the program handles these wrong values.	The program will be considered: 100% reliable if there is no error that will make it crash during runtime. 80% of errors are caught 40% reliable if the only error checking is during initialization. 0% reliable if project

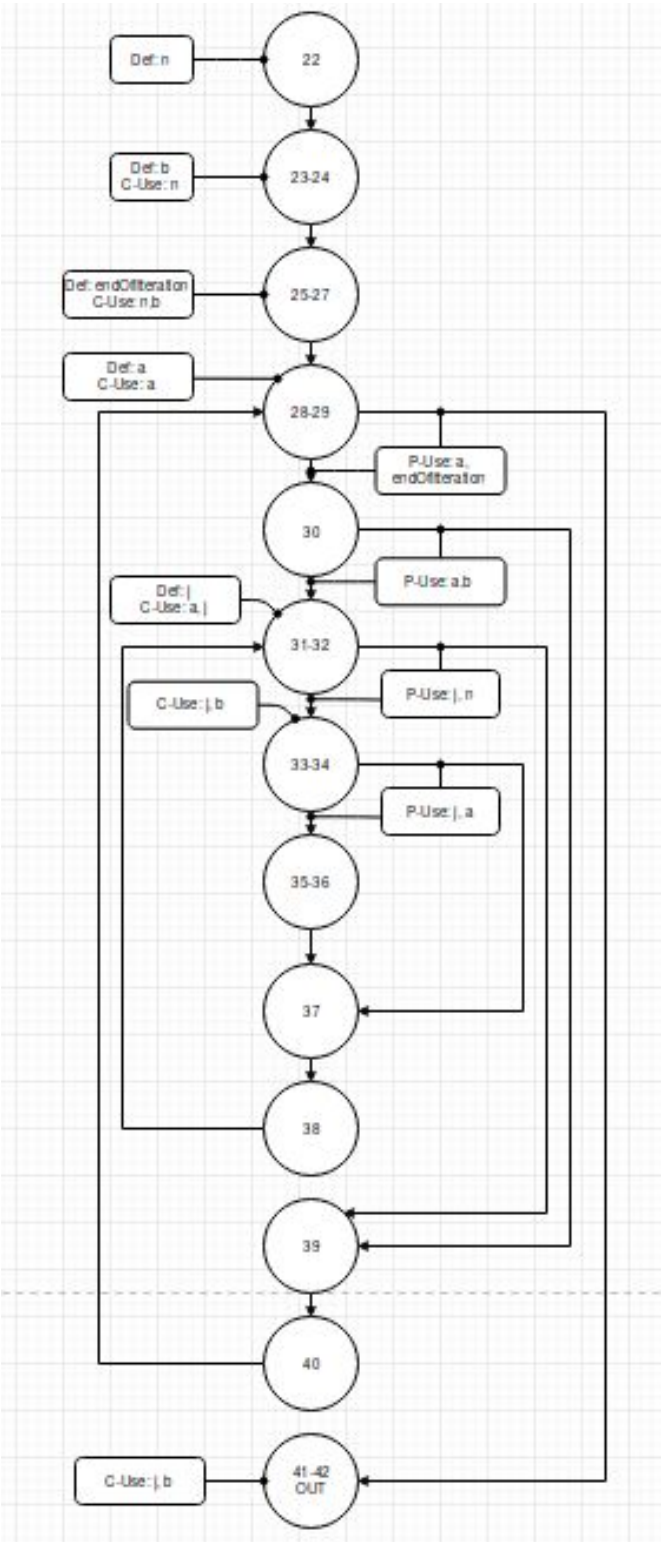
			does not compile/run.
2. Functionality	The program has a broad range of different operations that serve the base purpose.	Testing mutable users and asking them how they felt about the program.	This program has functionality if a user is satisfied with all the results they are given.

167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.

194. 3. Data-Flow Annotated Control Flow Graphs

a. **3.1 genEratosthenes**

b. **CFG:**



c.

d.

Node	**def**	**C-use**
22	{n}	∅
23-24	{b}	{n}
25-27	{endoflitteration}	{n,b}
28-29	{a}	{a}
30	∅	∅
31-32	{j}	{a,j}
33-34	∅	∅
35-36	∅	∅
37	∅	∅
38	∅	∅
39	∅	∅
40	∅	∅
41-42	{j}	{b}

e.

Edge	**P-use**
(28-29,41-42OUT)	**{a,** endoflitteration**}**
(28-29,30)	**{a,** endoflitteration**}**
(30,39)	**{a,b}**
(30,31-32)	**{a,b}**
(31-32,33-34)	**{j,n}**

(31-32,39)	**{j,n}**
(33-34,35-36)	**{j,a}**
(33-34,37)	**{j,a}**

f.

Var	**Node**	**DCU**	**DPU**
n	22	{23-24,25-27}	{(31-32,33-34),(31-32,39)}
b	23-26	{25-27,33-34,41-42}	{(30,39),(30,31-32)}
endoflitteration	25-27	∅	{(28-29,41-42),(28-29,30)}
a	28-29	{28-29,31-32}	{(28-29,41-42),(28-29,30),(30,39),(30,31-32),(33-34,35-36),(33-34,37)}
j	31-32	{31-32,33-34}	∅
j	41-42	∅	{(31-32,33-34),(31-32,39),(33-34,35-36),(33-34,37)}

g.

h.

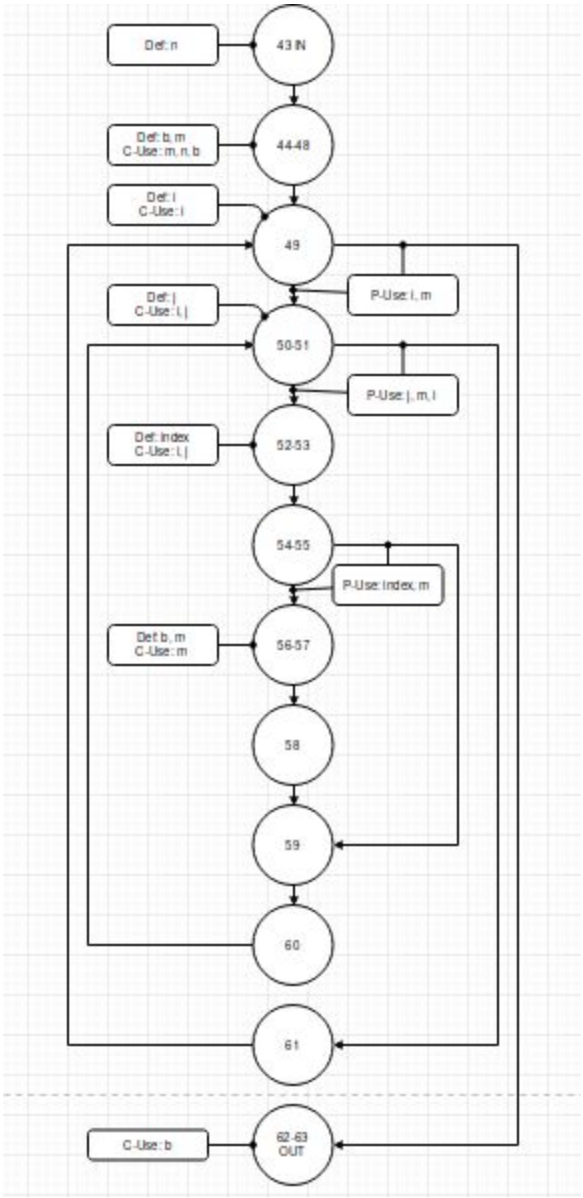
i.

j.

k.

l. 3.2 genSundaram

CFG:



m.

Node	Def	C-Use
43in	{n}	∅
44-48	{b,m}	{n,m,b}

49	{i}	{i}
50-51	{J}	{J}
52-53	{index}	{i,j}
54-55	∅	∅
56-57	{b,m}	{m}
58	∅	∅
59	∅	∅
60	∅	∅
61	∅	∅
62-63out	∅	{b}

n.

o.

Edge	**P-Use**
(49,50-51)	{i,m}
(49,62-63)	{i,m}
(50-51,52-53)	{J,m,i}
(50-51,61)	{J,m,i}
(54-55,56-57)	{index,m}
(54-55,59)	{index,m}

p.

q.

Var	**Node**	**DCU**	**DPU**
n	43in	{44-48}	∅

b	44-48	{44-48}	∅
m	44-48	{(44-48),(49,50-51),(49,62-65)}	{(49,50-51),(49,62-63),(50-51,52-53),(50-51,61),(54-55,56-57),(54-55,59)}
i	49	{(49,52-53),(49,50-51),(49,62-63)}	{(49,50-51),(49,62-63),(50-51,61),(50-51,52-53)}
J	50-51	{(50-51,52-53)}	{(50-51,61),(50-51,52-53)}
index	52-53	∅	{(54-55,56-57),(54-55,59)}
b	56-57	{62-63out}	∅
m	56-57	{56-57}	∅

r.

s.

t.

u.

v.

w.

x.

y.

z.

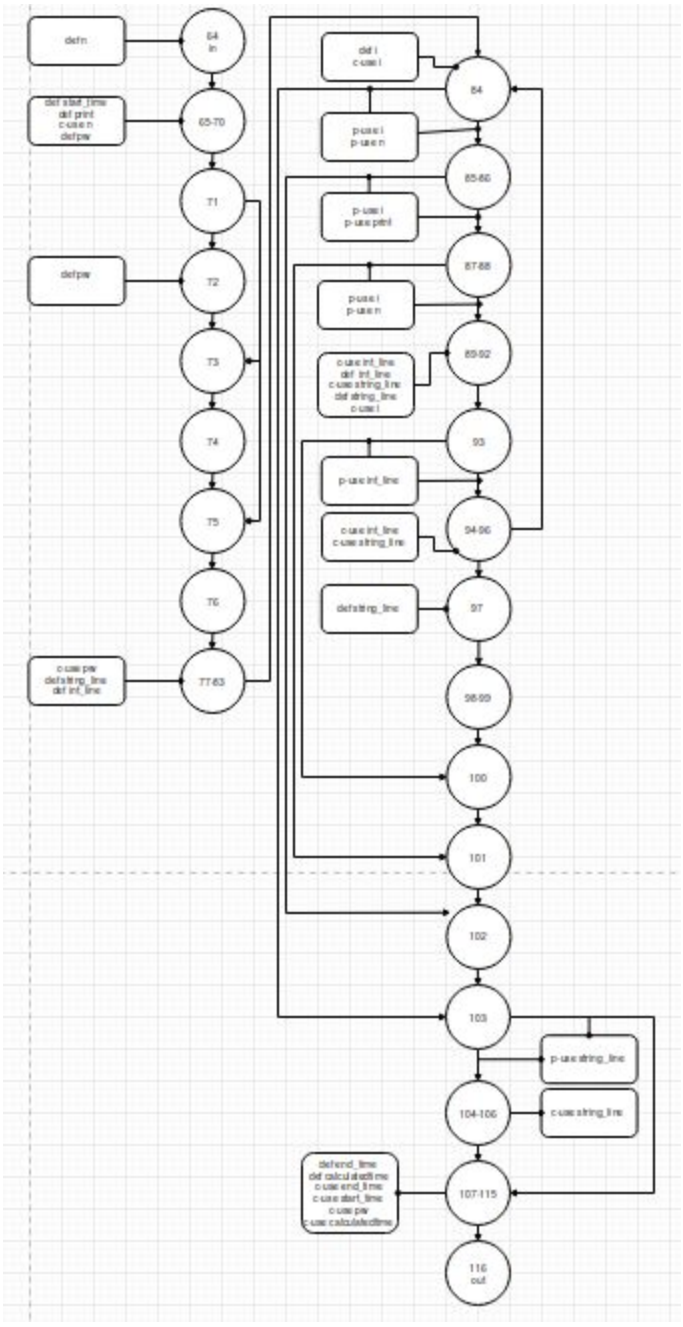
aa.

bb.

cc.

dd.3.3 fileConstructorSundaram

CFG:



ee.

Node	Def	C-Use
64in	{n}	∅
65-70	{start_time,print,pw}	∅

70	∅	∅
71	∅	∅
72	{pw}	∅
73	∅	∅
74	∅	∅
75	∅	∅
76	∅	∅
77-83	{string_line, int_line}	{pw}
84	{i}	{i}
88	∅	∅
89-92	{int_line,string_line}	{int_line,string_line,i}
93	∅	∅
94-96	∅	{int_line, string_line}
97	{string_line}	∅
98-99	∅	∅
100	∅	∅
101	∅	∅
102	∅	∅
103	∅	∅
104-106	∅	{string_line}
Node	**Def**	**C-Use**
107-115	{end_time,calculated_ti	{end_time,start_time,p

	me}	w,calculated_time}
116out	∅	∅

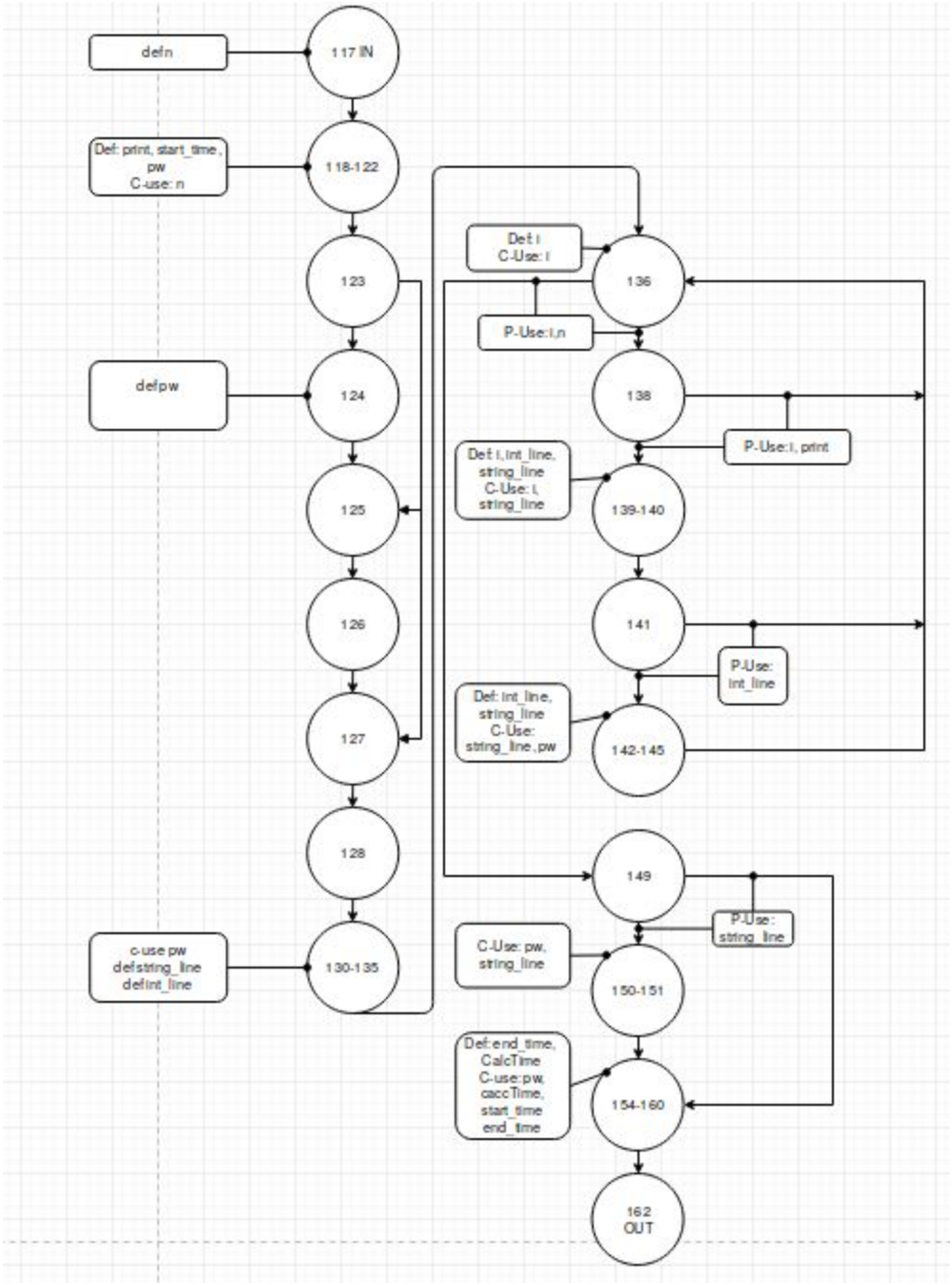
ff.

Edge	**P-Use**
(84,85-86)	{i,n}
(84,103)	{i,n}
(85-86,87-88)	{i,print}
(85-86,102)	{i,print}
(87-88,89-92)	{i,n}
(87-88,101)	{i,n}
(93,94-97)	{int_line}
(93-100)	{int_line}
(103,107-115)	{string_line}
(103,104-116)	{string_line}

Var	**Node**	**DCU**	**DPU**
n	64in	{65-70}	{(84,85-86),(84,103),(87-88,89-92),(87-88,101)}
start_time	65-70	{107-115}	∅
print	65-70	∅	{(85-86,87-88),(85-86,102)}
pw	65-70	∅	∅
pw	72	{72-83,107-115}	∅
string_line	77-83	∅	∅

int_line	77-83	∅	∅
i	84	{84,89-92}	{(84,85-86),(84,103),(85-86,87-88),(85-86,102),(87-88,89-92),(87-88,101)}
int_line	87-92	{87-92,94-96}	{(93,94-97),(93,100)}
string_line	89-92	{89-92,94-96}	∅
string_line	97	{104-106}	{(103,107-115),(103,104-116)}
end_time	107-115	{107-115}	∅
calculated_time	107-115	{107-115}	∅

gg. **3.4 fileConstructorEratosthenes**



195.

196.

197.

198.

Node	**Def**	**C-Use**
117in	{n}	∅
118-122	{print,start_time,pw}	{n}
123	∅	∅
124	{pw}	∅
125	∅	∅
126	∅	∅
127	∅	∅
128	∅	∅
130-135	{string_line, int_line}	{pw}
136	{i}	{i}
138	∅	∅
139-140	{int_line,string_line}	{i,string_line}
142-145	{int_line,string_line}	{string_line,pw}
149	∅	∅
150-151	∅	{pw,string_line}
154-160	{end_time,calculated_time}	{pw,calculated_time,start_time,end_time}
162out	∅	∅

Edge	P-use
(136,138)	{i,n}
(136,149)	{i,n}

(138,136)	{i,print}
(138,139-140)	{i,print}
(141,136)	{int_line}
(141,142-145)	{int_line}
(149,159-160)	{string_line}
(149,150-151)	{string_line}

Var	**Node**	**DCU**	**DPU**
n	117in	{118-122}	{(136,138),(136,149)}
print	118-122	∅	{(136,138),(136,139-140)}
start_line	118-122	{154-160}	∅
pw	118-122	∅	∅
pw	124	{142-145,150-151,154-160}	∅
string_line	130-135	{142-145}	∅
int_line	130-135	∅	∅
i	136	{136}	{(136,138),(136,149),(138,136),(138,139-140)}
int_line	139-140	∅	{(141,136),(141,142-145)}
string_line	139-140	{139-140}	∅
i	139-140	{139-140}	∅
int_time	142-145	∅	∅

string_line	142-145	{139-140}	∅
i	139-140	{139-140}	∅
end_time	159-160	{159-160}	∅
caculate_time	159-160	{159-160}	∅

199.

200. 4.Specification Based Testing

(Note: all actual results are based on the fact that java has certain exceptions when dealing with certain data types
following methods are being tested:
3.1 genEratosthenes(n)
3.2 genSundaram(m)
3.3 fileConstructorSundaram(l)
3.4 fileConstructorEratosthenes(k)

)

Parameter	Equivalence Class		Representative
	ID	Description	
n	1.1	All Positive Int's	1000
n	1.a	All Negative Int's	-1000
n	1.b	When n == NULL	NULL
n	1.c	When n is a String	"string"
n	1.d	When n is a float	42.0
n	1.2	When n == 0	0

Parameter	Equivalence Class		Representative
	ID	Description	
m	2.1	All Positive Int's	1000
m	2.a	All Negative Int's	-1000
m	2.b	When n == NULL	NULL
m	2.c	When n is a String	"string"
m	2.d	When n is a float	42.0
m	2.2	When n == 0	0

Parameter	Equivalence Class		Representative
	ID	Description	
l	3.1	All Positive Int's	1000

l	3.a All Negative Int's	-1000
l	3.b When n == NULL	NULL
l	3.c When n is a String	"is a string"
l	3.d When n is a float	42.0
l	3.2 When n == 0	0

Parameter	**Equivalence Class**		Representative
	ID	Description	
k	4.1	All Positive Int's	1000
k	4.a	All Negative Int's	-1000
k	4.b	When n == NULL	NULL
k	4.c	When n is a String	"string"
k	4.d	When n is a float	42.0
k	4.2	When n == 0	0

Parameter	Boundary Values			Test Case ID(s)
n	(Bound) - 1	Bound	(Bound) +1	TC: 1-6
	-1	0	1	
	2147483646	2147483647	2147483648	
	-2147483649	-2147483648	-2147483647	

Test Case ID	**n**	**Exp. Result**	**Actual. Result**

TC-1	MAX_INT	Generates all prime numbers between 0 and 2147483647	Generates all prime numbers between 0 and 2147483647
TC-2	MAX_INT +1	Overflow error " Could not compile"	Fail
TC-3	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-4	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-5	-1	Fail	Fail
TC-6	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-7	"string"	Fail	Fail
TC-8	42.0	Fail	Fail
TC-9	NULL	Fail	Fail
TC-10	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-11	-1000	Fail	Fail

Test Case	**m**	**exp result**	**actual result**
TC-12	MAX_INT	Generates all prime numbers between 0 and 2147483647	Generates all prime numbers between 0 and 2147483647
TC-13	MAX_INT +1	Overflow error " Could not compile"	Fail
TC-14	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646

TC-15	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-16	-1	Fail	Fail
TC-17	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-18	“string”	Fail	Fail
TC-19	42.0	Fail	Fail
TC-20	NULL	Fail	Fail
TC-21	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-22	-1000	Fail	Fail

Test Case	**I**	**exp result**	**actual result**
TC-23	MAX_INT	Generates all prime numbers between 0 and 2147483647	Generates all prime numbers between 0 and 2147483647
TC-24	MAX_INT +1	Overflow error “ Could not compile”	Fail
TC-25	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-26	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-27	-1	Fail	Fail
TC-28	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-29	“string”	Fail	Fail

TC-30	42.0	Fail	Fail
TC-31	NULL	Fail	Fail
TC-32	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-33	-1000	Fail	Fail

Test Case	**K**	**exp result**	**actual result**
TC-34	MAX_INT	Generates all prime numbers between 0 and 2147483647	Generates all prime numbers between 0 and 2147483647
TC-35	MAX_INT +1	Overflow error “ Could not compile”	Fail
TC-36	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-37	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-38	-1	Fail	Fail
TC-39	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-40	“string”	Fail	Fail
TC-41	42.0	Fail	Fail
TC-42	NULL	Fail	Fail
TC-43	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000

TC-44	-1000	Fail	Fail

We were able to reuse test cases TC-1 to TC-44 in order to fulfill Statement Testing, Branch Testing, Minimum Multiple Condition Testing, and Path coverage. We are confident that we have tested each variable in a way that has sufficiently shown proper coverage. We did have to add additional test cases to cover the Loop Test. This is shown below in Section 4.1.

201. 4.1 loop test

Test Case ID	Parameter(n)	Coverage	
		path	%
TC-45	0	22IN,23-24,25-27,28-29,41-42OUT	1/3
TC-46	1	22IN,23-24,25-27,28-29,30,39,40,28-29,41-42OUT	1/3
TC-47	2	22IN,23-24,25-27,28-29,30,(31-32,33-34,35-36,37,38)^2,39,40,28-29,41-42OUT	1/3

Test Case ID	Parameter(m)	Coverage	
		path	%
TC-48	0	43IN,44-48,49,62-63OUT	1/3
TC-49	1	43IN,44-48,49,50-51,52-53,54-55,56-57,58,59,60,50-51,61,49,62-63OUT	1/3
TC-50	2	43IN,44-48,(49,50-51,52-53,54-55,56-57,58,59,60,50-51,61)^2,62-63OUT	1/3

Test Case ID	Parameter(l)	Coverage	
		path	%
TC-51	0	64IN,65-83,84,103,107-114,115OUT	1/3
TC-52	2	64IN,65-83,84,86,88,84,103,107-114,115OUT	1/3
TC-53	4	64IN,65-83,(84,86,88,89-92,93,94-97)^2,103,107-114,115OUT	1/3

Test Case ID	Parameter(k)	Coverage	
		path	%
TC-54	0	117IN,118-135,136,149,154-160,162OUT	1/3
TC-55	2	117IN,118-135,136,138,139-140,141,142-145,136,149,154-160,162OUT	1/3
TC-56	3	117IN,118-135,(136,138,139-140,141,142-145)^2,149,154-160,162OUT	1/3

202. 5. Class Test Strategy

The modality of our project is uni-modal, because there is a strict order in which the methods must be called in order for this program. The file constructor methods must be called prior to the generate sieve methods. This is why the modality of our system is uni-modal.

For our test strategy, we are going to diverge from Binder's Class Test Strategy. We are doing this because there are no:

- No Class Variables
- No Polymorphic Calls

    - Since our project only handles one class there is no possibility of a polymorphic call.
- No Flattened Class Scope
    - Following the same logic laid out above, since there are no inherited methods because there inheritance calls.

Our Test Strategy will consist of these testing standards in order:

1. Method Scope Testing
    a. Category Partition test
    b. Source-Code test
2. Class Scope Testing

We chose these methods of testing because they are the most efficient and carry the highest effectiveness with minimal effort.

203. 6. Object-Oriented Test Implementation Method Scope Test

Category Partition Test:
The methods that we are currently are as follows:

- genEratosthenes
    - Represented with the parameter n
- genSundaram
    - Represented with the parameter m

- fileConstructorSundaram
    - Represented with the parameter l
- fileConstructorEratosthenes
    - Represented with the parameter k

Parameter	Equivalence Class		Representative
	ID	Description	
n	1.1	All Positive Int's	1000
n	1.a	All Negative Int's	-1000
n	1.b	When n == NULL	NULL
n	1.c	When n is a String	"string"
n	1.d	When n is a float	42.0
n	1.2	When n == 0	0

Parameter	Equivalence Class		Representative
	ID	Description	
m	2.1	All Positive Int's	1000
m	2.a	All Negative Int's	-1000
m	2.b	When n == NULL	NULL
m	2.c	When n is a String	"string"
m	2.d	When n is a float	42.0
m	2.2	When n == 0	0

Parameter	Equivalence Class		Representative
	ID	Description	
l	3.1	All Positive Int's	1000
l	3.a	All Negative Int's	-1000
l	3.b	When n == NULL	NULL
l	3.c	When n is a String	"is a string"
l	3.d	When n is a float	42.0
l	3.2	When n == 0	0

Parameter	Equivalence Class		Representative
	ID	Description	
k	4.1	All Positive Int's	1000
k	4.a	All Negative Int's	-1000
k	4.b	When n == NULL	NULL
k	4.c	When n is a String	"string"
k	4.d	When n is a float	42.0
k	4.2	When n == 0	0

Parameter	Boundary Values			Test Case ID(s)
n	(Bound) - 1	Bound	(Bound) +1	TC: 1-6
	-1	0	1	
	2147483646	2147483647	2147483648	
	-2147483649	-2147483648	-2147483647	

Test Case ID	n	Exp. Result	Actual. Result
TC-1	10000	Generates all prime numbers between 0 and 10000	Generates all prime numbers between 0 and 10000
TC-2	MAX_INT +1	Overflow error “ Could not compile”	Fail
TC-3	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-4	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-5	-1	Fail	Fail
TC-6	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-7	“string”	Fail	Fail
TC-8	42.0	Fail	Fail
TC-9	NULL	Fail	Fail
TC-10	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-11	-1000	Fail	Fail

Test Case	m	exp result	actual result
TC-12	10000	Generates all prime numbers between 0 and 10000	Generates all prime numbers between 0 and 10000
TC-13	MAX_INT +1	Overflow error “ Could not compile”	Fail

TC-14	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-15	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-16	-1	Fail	Fail
TC-17	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-18	“string”	Fail	Fail
TC-19	42.0	Fail	Fail
TC-20	NULL	Fail	Fail
TC-21	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-22	-1000	Fail	Fail

Test Case	**I**	**exp result**	**actual result**
TC-23	4,200,000	Generates all prime numbers between 0 and 4,200,000	Generates all prime numbers between 0 and 4,200,000
TC-24	MAX_INT +1	Overflow error “ Could not compile”	Fail
TC-25	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-26	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-27	-1	Fail	Fail

TC-28	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-29	“string”	Fail	Fail
TC-30	42.0	Fail	Fail
TC-31	NULL	Fail	Fail
TC-32	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-33	-1000	Fail	Fail

Test Case	**K**	**exp result**	**actual result**
TC-34	4,200,000	Generates all prime numbers between 0 and 4,200,000	Generates all prime numbers between 0 and 4,200,000
TC-35	MAX_INT +1	Overflow error “ Could not compile”	Fail
TC-36	MAX_INT - 1	Generates all prime numbers between 0 and 2147483646	Generates all prime numbers between 0 and 2147483646
TC-37	0	Generates all primes between 0 and 0	Generates all primes between 0 and 0
TC-38	-1	Fail	Fail
TC-39	1	Generates all primes between 0 and 1	Generates all primes between 0 and 1
TC-40	“string”	Fail	Fail
TC-41	42.0	Fail	Fail
TC-42	NULL	Fail	Fail

TC-43	1000	Generates all prime numbers between 0 and 1000	Generates all prime numbers between 0 and 1000
TC-44	-1000	Fail	Fail

Source-Code Test:

The methods that we are currently are as follows:

- genEratosthenes
  - Represented with the parameter n
- genSundaram
  - Represented with the parameter m
- fileConstructorSundaram
  - Represented with the parameter l
- fileConstructorEratosthenes
  - Represented with the parameter k

Test Case ID	**Parameter(n)**	**Coverage**	
		path	**%**
TC-45	**0**	**22IN,23-24,25-27,28-29,41-42OUT**	**1/3**
TC-46	**1**	**22IN,23-24,25-27,28-29,30,39,40,28-29,41-42OUT**	**1/3**
TC-47	**2**	**22IN,23-24,25-27,28-29,30,(31-32,33-34,35-36,37,38)^2,39,40,28-29,41-42OUT**	**1/3**

Test Case ID	**Parameter(m)**	**Coverage**	
		path	**%**
TC-48	**0**	**43IN,44-48,49,62-63OUT**	**1/3**

TC-49	**1**	**43IN,44-48,49,50-51,52-53,54-55,56-57,58,59,60,50-51,61,49,62-63OUT**	**1/3**
TC-50	**2**	**43IN,44-48,(49,50-51,52-53,54-55,56-57,58,59,60,50-51,61)^2,62-63OUT**	**1/3**

Test Case ID	**Parameter(l)**	**Coverage**	
		path	**%**
TC-51	**0**	**64IN,65-83,84,103,107-114,115OUT**	**1/3**
TC-52	**2**	**64IN,65-83,84,86,88,84,103,107-114,115OUT**	**1/3**
TC-53	**4**	**64IN,65-83,(84,86,88,89-92,93,94-97)^2,103,107-114,115OUT**	**1/3**

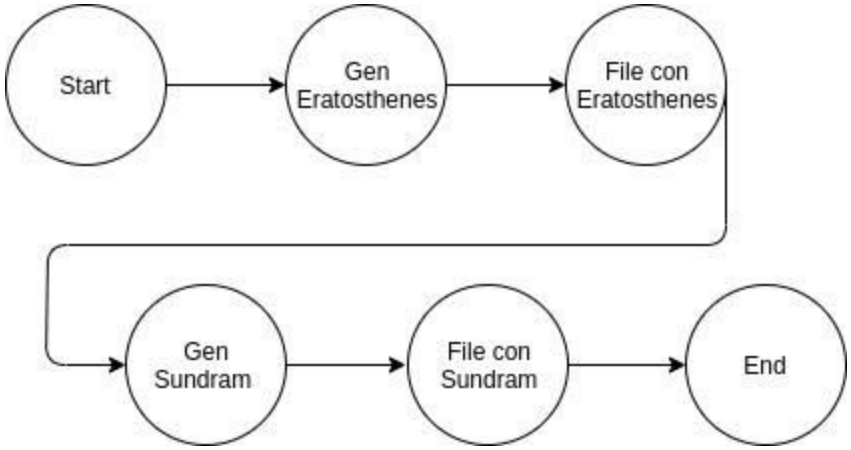
Test Case ID	**Parameter(k)**	**Coverage**	
		path	**%**
TC-54	**0**	**117IN,118-135,136,149,154-160,162OUT**	**1/3**
TC-55	**2**	**117IN,118-135,136,138,139-140,141,142-145,136,149,154-160,162OUT**	**1/3**
TC-56	**3**	**117IN,118-135,(136,138,139-140,141,142-145)^2,149,154-160,162OUT**	**1/3**

Polymorphism Test:

We don’t need to use this test methodology for the reasons stated in Section 5.

Class Scope Test

The order of the methods is as follows:



- fileConstructorSundaram, genSundgaram
- fileConstructorEratosthenes, genEratosthenes

Flattened Class Scope and Class Interaction Test

We don't use this test methodology for the reasons stated in Section 5.

204. 7. J-Unit Analysis

Most of the test cases that we expected to pass, executed correctly and have passed. Most of the test cases we expected to fail, either failed to execute or executed correctly but gave us a wrong result. We weren't able to test all of our test cases, unfortunately. The test cases that take in: MAX_INT, MAX_INT+1, and "string". The MAX_INT test cases were inefficient to test and thus deemed outside the scope of this project. The "string" test cases cannot be tested because they won't compile. Also, the largest number we were able to effectively test is 10,000.
The four test cases that didn't perform as expected were TC-24,TC-

205. 8. Appendix

206. 8.1 Code example

```
1. import java.util.Scanner;
2. import java.io.FileNotFoundException;
3. import java.io.PrintWriter;
4. import java.io.UnsupportedEncodingException;
5. import java.util.BitSet;
```

```
public class Main {
   public static void main(String[] args) {
         Alg a = new Alg();
         System.out.println("Hey, to what number do ya want?");
         //input the number, this will find primes up to that number using the two
         //different algorithms
         Scanner kb = new Scanner(System.in);
         int num = kb.nextInt();
         //reads input
         a.fileConstructorSundaram(num);
         System.out.println("\n\n\n---\n\n\n");
         a.fileConstructorEratosthenes(num);
         //generates the files and terminal output for both sets
   }
}
public class Alg {
   private BitSet genEratosthenes(int n) {
         //int n is the cap for prime numbers
         BitSet b = new BitSet(n);
         //bit sets for speed
         b.clear();
         double endOfIteration = Math.sqrt(n)  + 1;
         //stopping point of bitset iteration very early, the square root of n is the benchmark
         for (int a = 1; a < endOfIteration; a++) {
         if (b.get(a + 1) == false) {
               //checks to see if current number is unmarked, if it is it is prime
               for (int j = (a + 2); j < n + 1; j++) {
                     //if number is prime, all multiples of that number are flagged
                     if ((j % (a + 1)) == 0) {
                     b.set(j, true);
                     //flag
                     }
               }
         }
         }
         return b;
   }
   private BitSet genSundaram(int n){
         int m = ((n-2)/2);
         //endpoint number for this sundaram
         BitSet b = new BitSet(m);
         b.clear();
         //this algorithm flags all entries in the for loop using the formula i + j + 2ij
         for (int i = 1; i <= m + 1; i++){
         //i is less than or equal to the int m used above
         for(int j = i; j <= (m-i)/((2*i)+1) + 1; j++){
               //j is equal to i and is less a form that ((m-i)/((2*i)+1) +1) so it runs smoothly
```

```
                        int index = (i + j + 2 * i * j);
                        //bit array is flagged at i + j + 2ij
                        if (index <= m) {
                        //another check to ensure the algorithm
   does not go outside its bounds
                        b.set(index,  true);
                        //flag
                        }
            }
            }
            return b;
      }
      public void fileConstructorSundaram(int n) {
            long start_time = System.currentTimeMillis();
            //writes start time
            BitSet print = genSundaram(n);
            //generates sudram bitset
            PrintWriter pw = null;
            //file
            try {
            pw = new PrintWriter("primesSundaram.txt", "UTF-8");
            } catch (FileNotFoundException e) {
            e.printStackTrace();
            } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            }
            //file
            pw.println("Sundaram Solution:");
            System.out.println("Sundaram Solution:");
            String string_line = " 2";
            //sudram dose not account for 2 as a  prime
            int int_line = 1;
            for (int i = 1; i <= ((n-2)/2) + 1; i++) {
            //itterates through sudram bitset
            if (print.get(i) == false) {
                  //checks flag  in sudram set
                  if((2*i+1) <= n){
                        //checks if prime is out of bounds
                        int_line++;
                        string_line = string_line + " " + (2*i+1);
                        //does sudram calculation 2i+1
                        if (int_line > 10) {
                        int_line = 0;
                        pw.println(string_line);
                        System.out.println(string_line);
                        string_line = "";
                        }
                        //puts in nice lines of primes
                  }
                  }
            }
            if (!string_line.equals("")) {
                  pw.println(string_line);
```

```
105.                System.out.println(string_line);
106.          }
107.          //also for putting in nice lines
108.          long end_time = System.currentTimeMillis();
109.          //writes end  time
110.          String calculatedtime = "Whoah man, that took " +
   (end_time - start_time) + " milliseconds to execute...  Yikes!";
111.          //great joke
112.          pw.println(calculatedtime);
113.          System.out.println(calculatedtime);
114.          pw.close();
115.          //close file writer
116.          }
117.          public void  fileConstructorEratosthenes(int n) {
118.          long start_time = System.currentTimeMillis();
119.          //writes start time
120.          BitSet print = genEratosthenes(n);
121.          //gets erato bitset
122.          PrintWriter pw = null;
123.          try {
124.                pw = new PrintWriter("primesEratosthenes.txt",
   "UTF-8");
125.          } catch (FileNotFoundException e) {
126.                e.printStackTrace();
127.          } catch (UnsupportedEncodingException e) {
128.                e.printStackTrace();
129.          }
130.          //for file
131.          System.out.println("Eratosthenes Solution:");
132.          pw.println("Eratosthenes Solution:");
133.          String string_line = "";
134.          //makes it "pretty"
135.          int int_line = 0;
136.          for (int i = 2; i <= n; i++) {
137.                //itterate thrugh bitset and get primes
138.                if (print.get(i) == false) {
139.                int_line++;
140.                string_line = string_line + " " + i;
141.                if (int_line > 10) {
142.                      int_line = 0;
143.                      pw.println(string_line);
144.                      System.out.println(string_line);
145.                      string_line = "";
146.                }
147.                }
148.          }
149.          if (!string_line.equals("")) {
150.                pw.println(string_line);
151.                System.out.println(string_line);
152.          }
153.          //makes nice lines
154.          long end_time = System.currentTimeMillis();
155.          //writes end time
```

```
207.          String calcTime = "Whoah man, that took " + (end_time -
   start_time) + " milliseconds to execute...  Yikes!";
157.          //HAHAHAHAH
158.          pw.println(calcTime);
159.          System.out.println(calcTime);
160.          pw.close();
161.          }
162.    }
```