TKINTER TUTORIAL

Xiaofeng Gao
Dept. of Computer Science
Shanghai Jiao Tong University

Outline

- Introduction to Tkinter
- Case Study: Sorting Network

Introduction to Tkinter

Introduction to Tkinter

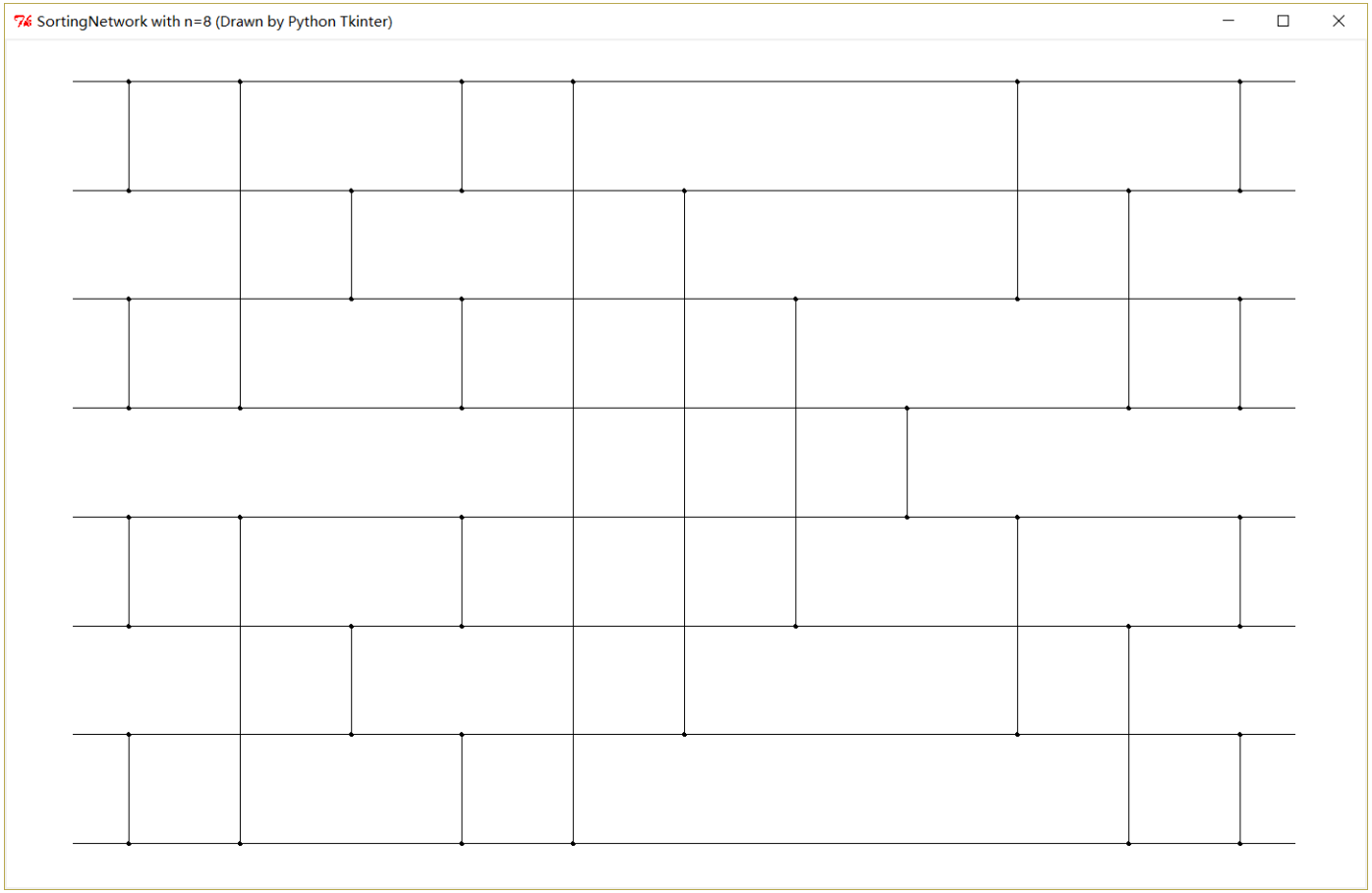
- *Tkinter* is the standard Python interface to the *Tk* GUI toolkit, and is Python's standard Graphical User Interface.
- It is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.
- The name *Tkinter* comes from *Tk* interface.
- https://docs.python.org/2/library/tkinter.html

A Simple Hello World Program

```
from Tkinter import *


class Application(Frame):
    def say_hi(self):
        print "hi there, everyone!"

    def createWidgets(self):
        self.QUIT = Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"] = "red"
        self.QUIT["command"] = self.quit

        self.QUIT.pack({"side": "left"})

        self.hi_there = Button(self)
        self.hi_there["text"] = "Hello",
        self.hi_there["command"] = self.say_hi

        self.hi_there.pack({"side": "left"})

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()


root = Tk()
app = Application(master=root)
app.mainloop()
root.destroy()
```

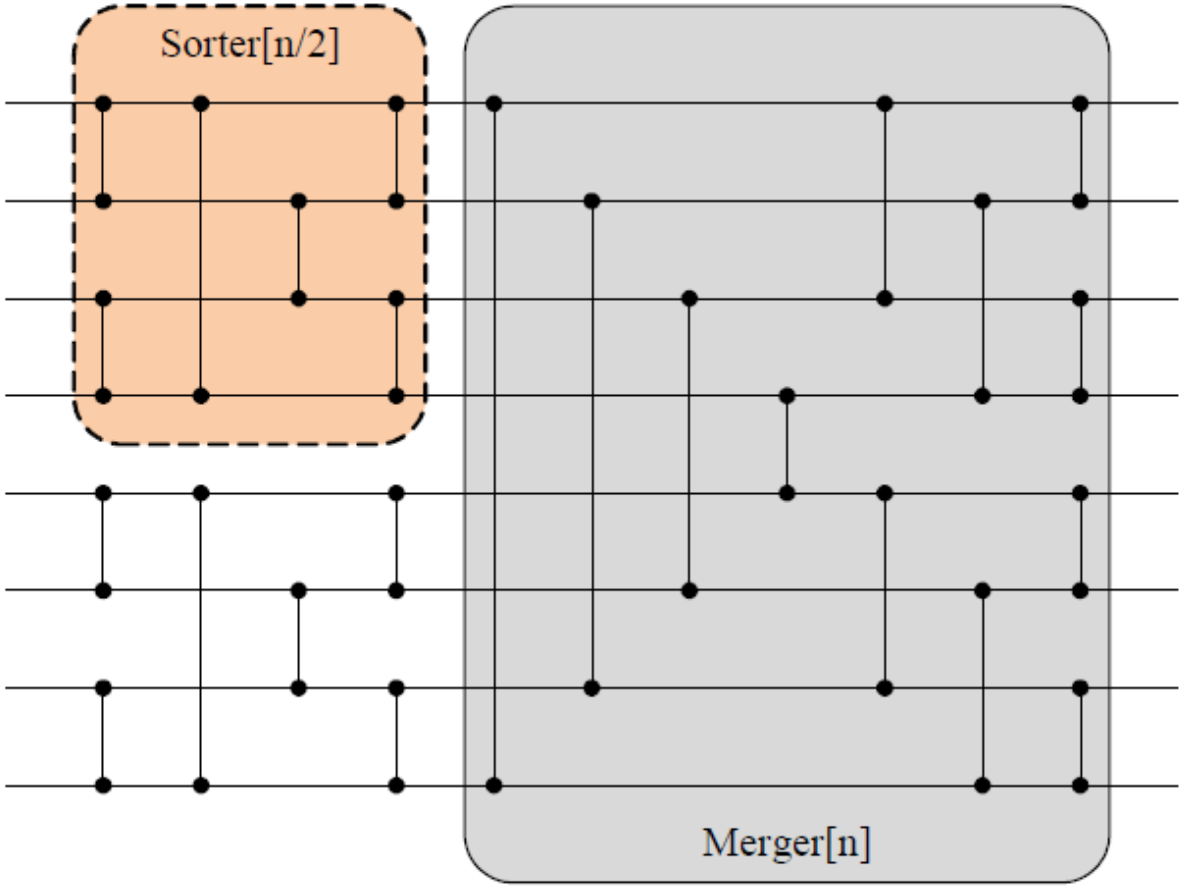
Case Study: Sorting Network

Problem Analysis



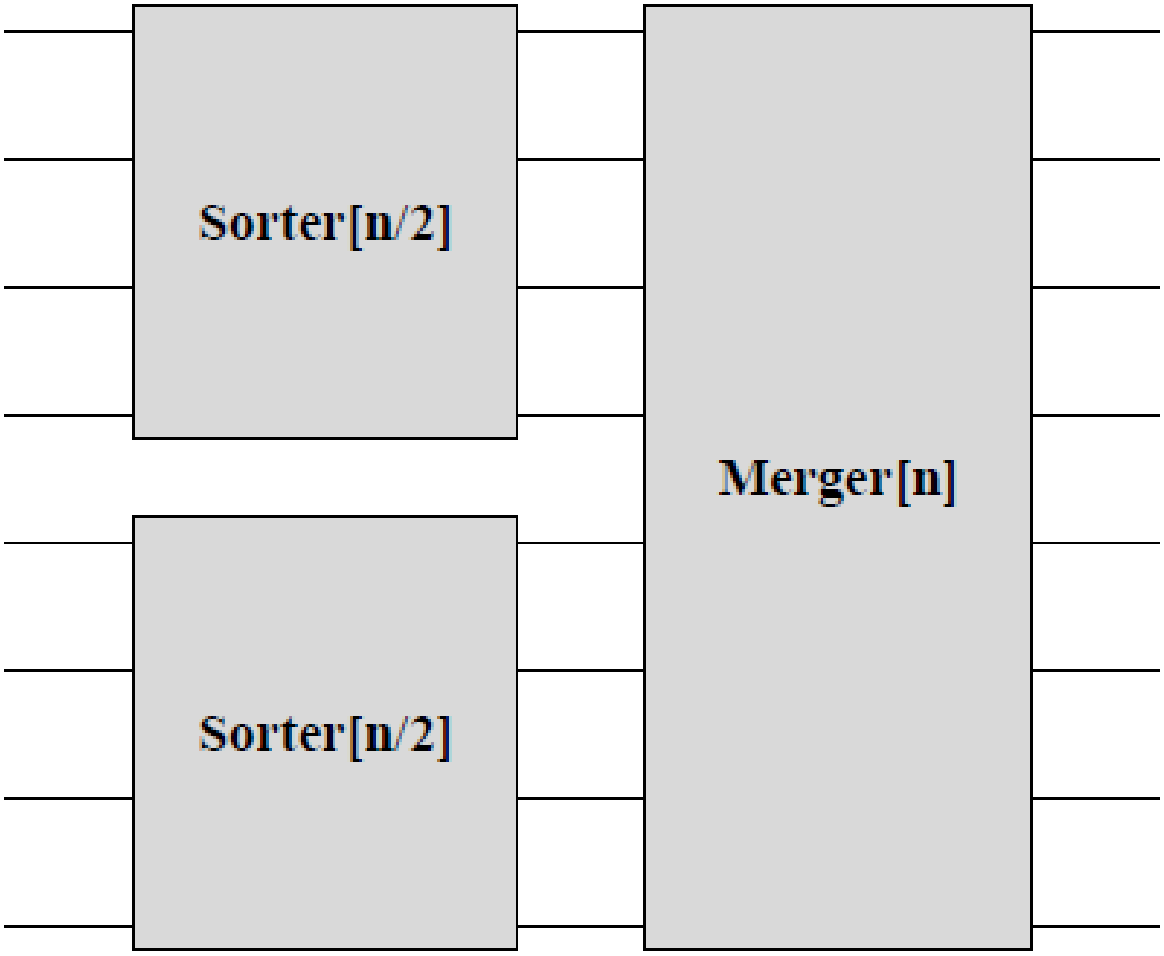
Sorting Network with $n=2^3=8$

Problem Analysis



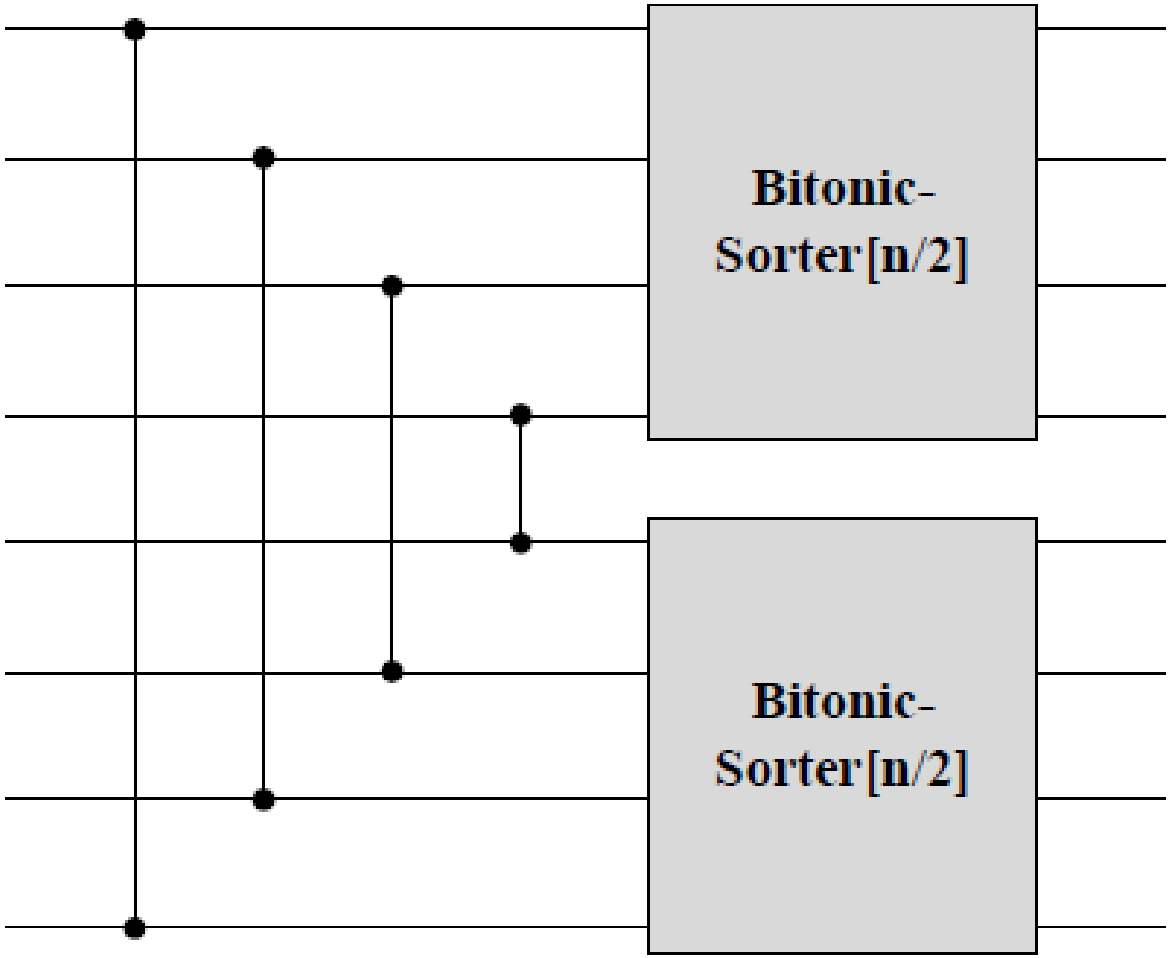
Sorting Network with $n=2^3=8$

Problem Analysis



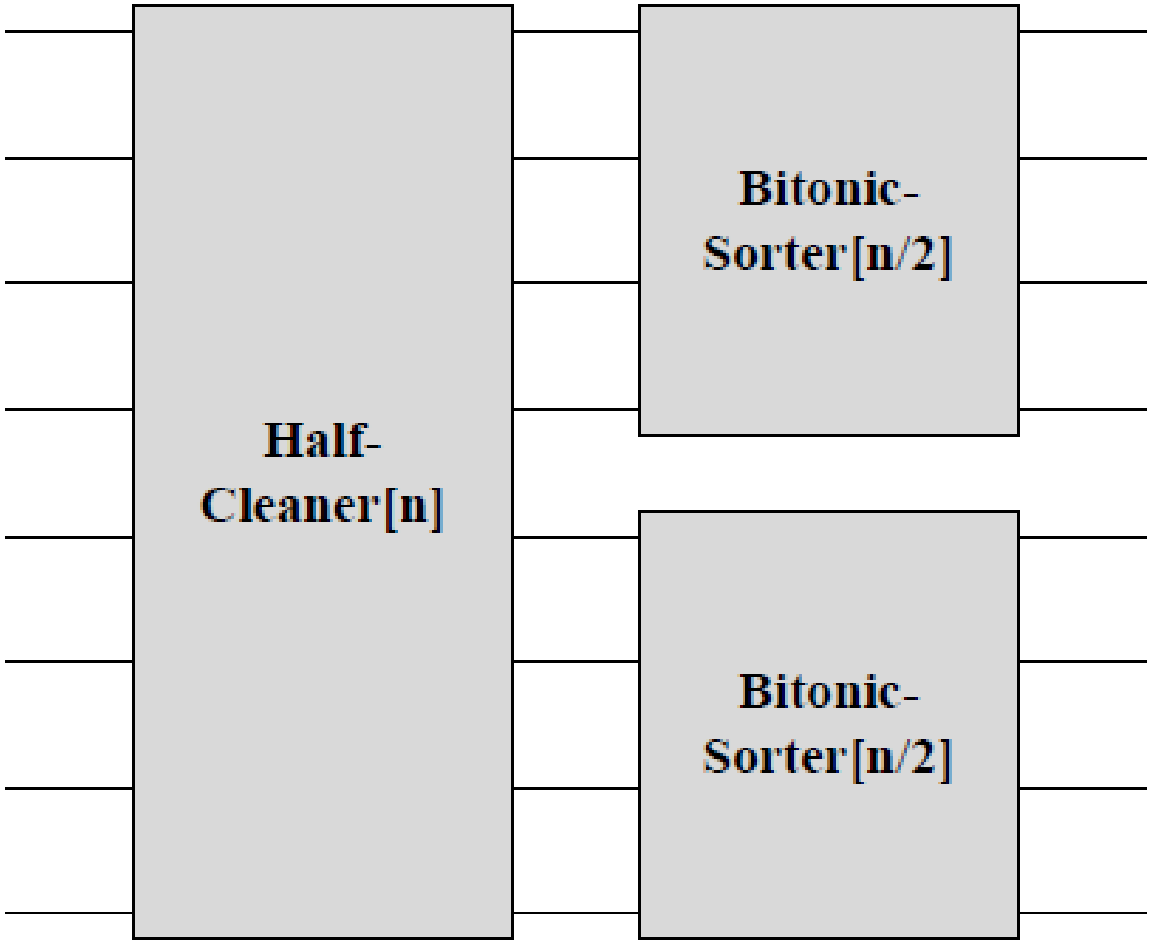
Sorter [n]

Problem Analysis



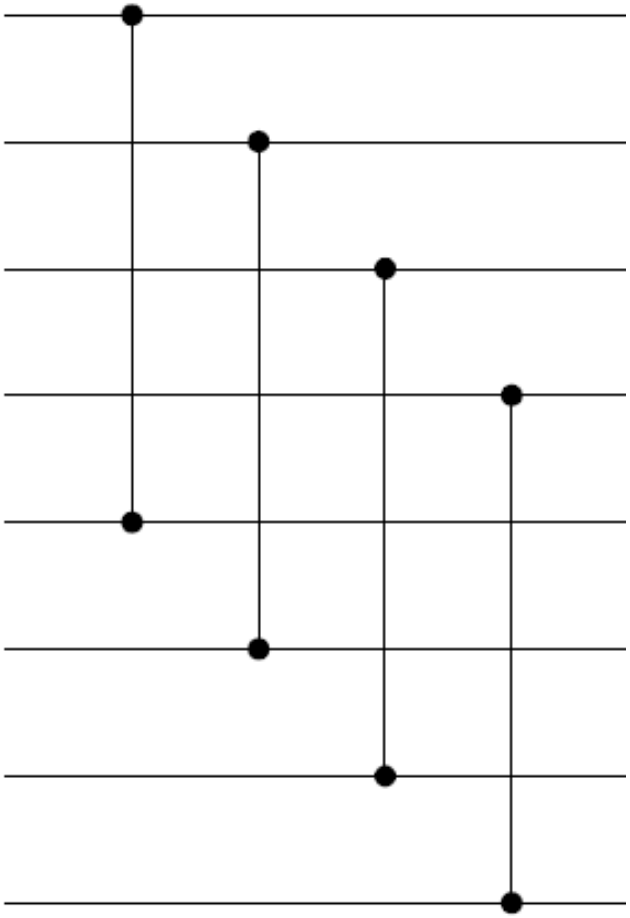
Merger [n]

Problem Analysis



Bitonic-Sorter [n]

Problem Analysis



Half-Cleaner [n]

Problem Analysis

- Sorter [n]
  - 2 * Sorter [n/2]
    - Defined recursively
  - Merger [n]
    - A triangle-shaped network
    - 2 * Bitonic-Sorter [n/2]
      - Half-Cleaner [n/2]
        - » A parallelogram-shaped network
      - 2 * Bitonic-Sorter [n/4]
        - » Defined recursively

Implementation

- Preliminary

```
class MyCanvas(Canvas):
    def __init__(self, master, hLineWidth=1, vLineWidth=1, radius=2, **kwargs):
        Canvas.__init__(self, master, kwargs)
        self.hLineWidth = hLineWidth
        self.vLineWidth = vLineWidth
        self.radius = radius

    def create_segment_h(self, x, y, l):
        self.create_line(x, y, x + l, y, width=self.hLineWidth)
        self.create_oval(x - self.radius, y - self.radius, x + self.radius, y + self.radius, fill='black')
        self.create_oval(x + l - self.radius, y - self.radius, x + l - self.radius, y + self.radius, fill='black')

    def create_segment_v(self, x, y, l):
        self.create_line(x, y, x, y + l, width=self.vLineWidth)
        self.create_oval(x - self.radius, y - self.radius, x + self.radius, y + self.radius, fill='black')
        self.create_oval(x - self.radius, y + l - self.radius, x + self.radius, y + l + self.radius, fill='black')

    def create_line_h(self, x, y, l):
        self.create_line(x, y, x + l, y, width=self.hLineWidth)

    def create_line_v(self, x, y, l):
        self.create_line(x, y, x, y + l, width=self.vLineWidth)
```

Problem Analysis

- Sorter [n]
  - 2 * Sorter [n/2]
    - Defined recursively
  - Merger [n]
    - A triangle-shaped network
    - 2 * Bitonic-Sorter [n/2]
      - Half-Cleaner [n/2]
        - » A parallelogram-shaped network
      - 2 * Bitonic-Sorter [n/4]
        - » Defined recursively

Implementation

- Half-Cleaner

```
class HalfCleaner:
    def __init__(self, size):
        self.size = size

    def hNum(self):
        return self.size / 2 if self.size > 1 else 0

    def draw(self, cvs, x, y, hScale, vScale):
        if self.size > 1:
            for i in range(self.size):
                cvs.create_line_h(x, y + i * vScale, self.hNum() * hScale)
            for i in range(self.size / 2):
                cvs.create_segment_v(x + (i + 0.5) * hScale, y + i * vScale, (self.size / 2) * vScale)
```

Problem Analysis

- Sorter [n]
  - 2 * Sorter [n/2]
    - Defined recursively
  - Merger [n]
    - A triangle-shaped network
    - 2 * Bitonic-Sorter [n/2]
      - Half-Cleaner [n/2]
        - » A parallelogram-shaped network
      - 2 * Bitonic-Sorter [n/4]
        - » Defined recursively

Implementation

- Bitonic-Sorter

```
class BitonicSorter:
    def __init__(self, size):
        self.size = size
        if self.size > 1:
            self.halfCleaner = HalfCleaner(self.size)
            self.subSorter = BitonicSorter(self.size / 2)

    def hNum(self):
        return self.halfCleaner.hNum() + self.subSorter.hNum() if self.size > 1 else 0

    def draw(self, cvs, x, y, hScale, vScale):
        if self.size > 1:
            self.halfCleaner.draw(cvs, x, y, hScale, vScale)
            self.subSorter.draw(cvs, x + self.halfCleaner.hNum() * hScale, y, hScale, vScale)
            self.subSorter.draw(cvs, x + self.halfCleaner.hNum() * hScale, y + (self.size / 2) * vScale, hScale, vScale)
```

Problem Analysis

- Sorter [n]
  - 2 * Sorter [n/2]
    - Defined recursively
  - Merger [n]
    - A triangle-shaped network
    - 2 * Bitonic-Sorter [n/2]
      - Half-Cleaner [n/2]
        - » A parallelogram-shaped network
      - 2 * Bitonic-Sorter [n/4]
        - » Defined recursively

Implementation

- Merger

```
class Merger:
    def __init__(self, size):
        self.size = size
        if size > 1:
            self.subSorter = BitonicSorter(size / 2)

    def hNum(self):
        return self.size / 2 + self.subSorter.hNum() if self.size > 1 else 0

    def draw(self, cvs, x, y, hScale, vScale):
        if self.size > 1:
            for i in range(self.size):
                cvs.create_line_h(x, y + i * vScale, (self.size / 2) * hScale)
            for i in range(self.size / 2):
                cvs.create_segment_v(x + (i + 0.5) * hScale, y + i * vScale, (self.size - 1 - 2 * i) * vScale)
            self.subSorter.draw(cvs, x + (self.size / 2) * hScale, y, hScale, vScale)
            self.subSorter.draw(cvs, x + (self.size / 2) * hScale, y + (self.size / 2) * vScale, hScale, vScale)
```

Problem Analysis

- Sorter [n]
  - 2 * Sorter [n/2]
    - Defined recursively
  - Merger [n]
    - A triangle-shaped network
    - 2 * Bitonic-Sorter [n/2]
      - Half-Cleaner [n/2]
        - » A parallelogram-shaped network
      - 2 * Bitonic-Sorter [n/4]
        - » Defined recursively

Implementation

- Sorter

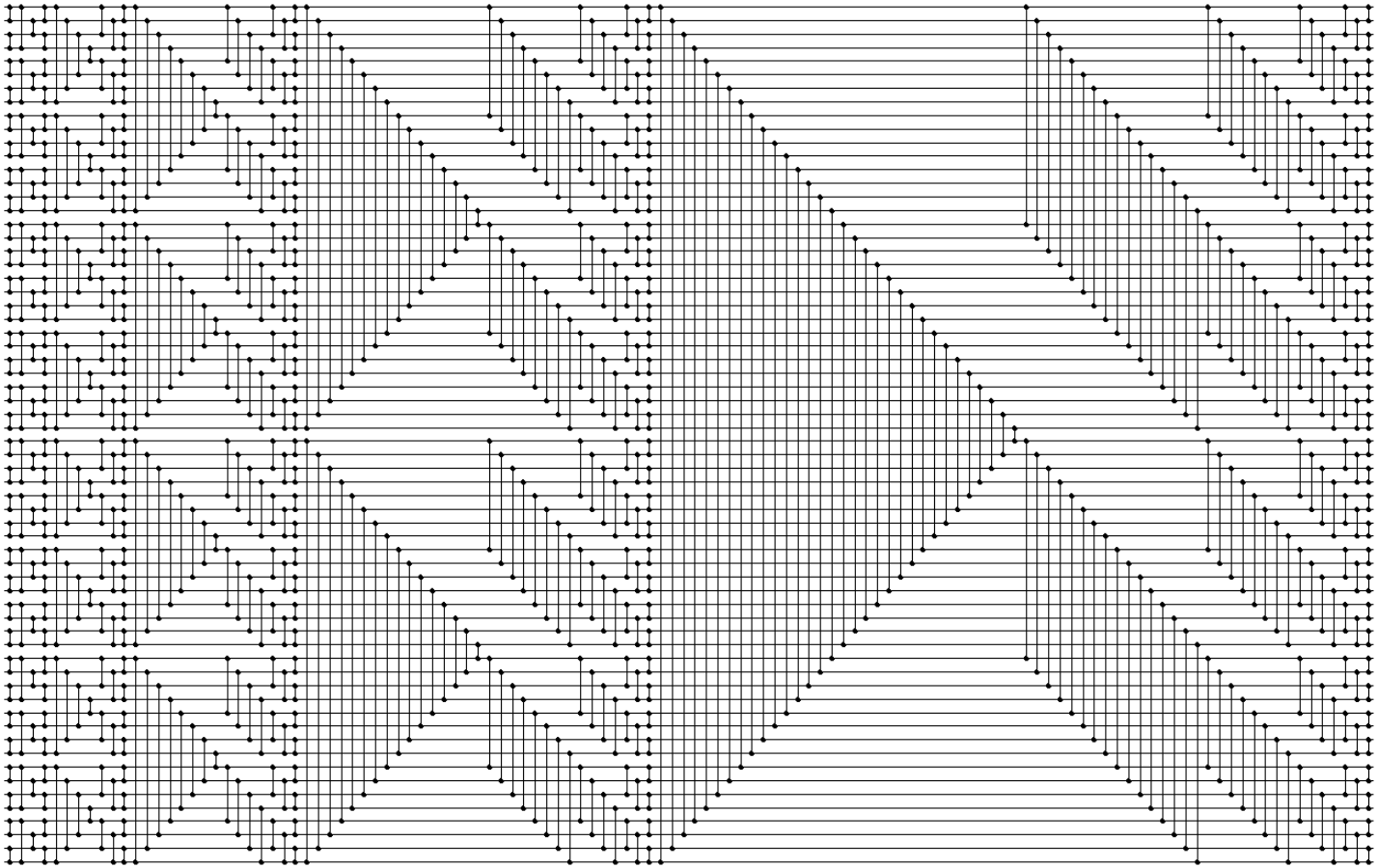
```
class Sorter:
    def __init__(self, size):
        self.size = size
        if size > 1:
            self.subSorter = Sorter(size / 2)
            self.merger = Merger(size)

    def hNum(self):
        return self.subSorter.hNum() + self.merger.hNum() if self.size > 1 else 0

    def draw(self, cvs, x, y, hScale, vScale):
        if self.size > 1:
            self.subSorter.draw(cvs, x, y, hScale, vScale)
            self.subSorter.draw(cvs, x, y + (self.size / 2) * vScale, hScale, vScale)
            self.merger.draw(cvs, x + self.subSorter.hNum() * hScale, y, hScale, vScale)
```

Implementation

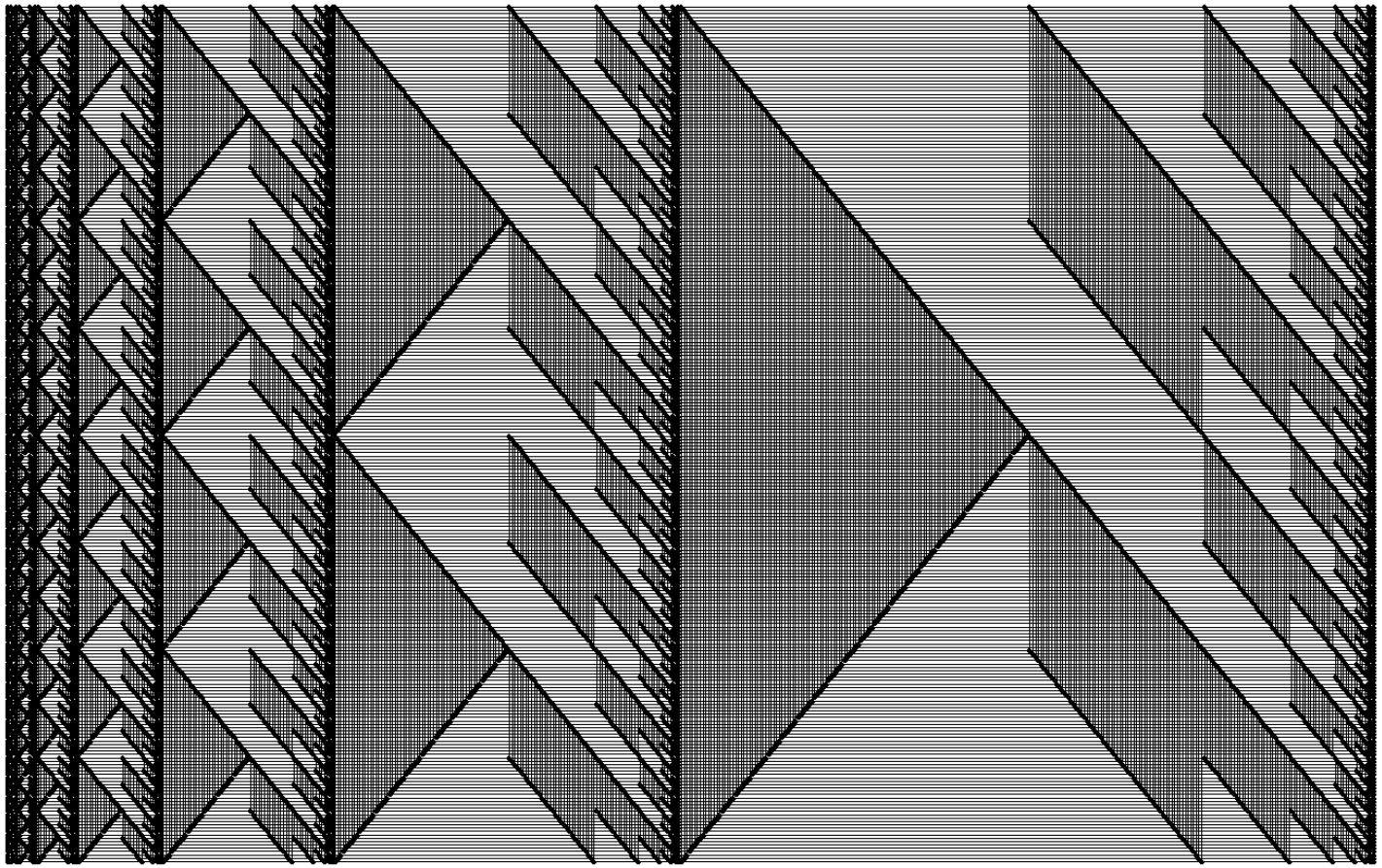
- Main Function

```
if __name__ == '__main__':
    k = input('please input the number k: ')
    n = 2 ** k
    sortingNetwork = Sorter(n)

    winW, winH = 2400 * 0.6, 1500 * 0.6
    hMargin, vMargin = winW / 20, winH / 20
    hScale, vScale = (winW - 2 * hMargin) / sortingNetwork.hNum(), (winH - 2 * vMargin) / (n - 1)

    root = Tk()
    root.title('Sorting Network with n=%d (Drawn by Python Tkinter)' % n)
    cvs = MyCanvas(root, bg='white', width=winW, height=winH)
    sortingNetwork.draw(cvs, hMargin, vMargin, hScale, vScale)
    cvs.pack()
    root.mainloop()
```

Result ($n=2^6=64$)



Result ($n=2^8=256$)



Exercise (Bonus)

3. **Transposition Sorting Network**: A comparison network is a **transposition network** if each comparator connects adjacent lines, as in the network in Fig. 1.

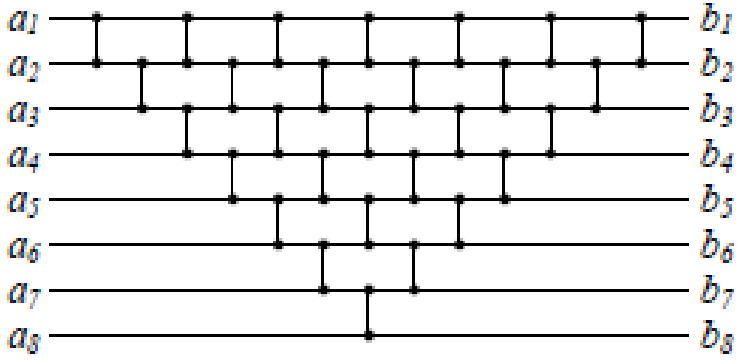


Figure 1: A Transposition Network Example

(a) Prove that a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \cdots, 1\rangle$. (Hint: Use an induction argument analogous to the *Domain Conversion Lemma*.)

(b) (Bonus) Given any $n \in \mathbb{N}$, write a program using Tkinter in Python to draw a figure similar to Fig. 1 with n input wires.

THANK YOU!