traitement données de fork selon différentes définitions

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np
```

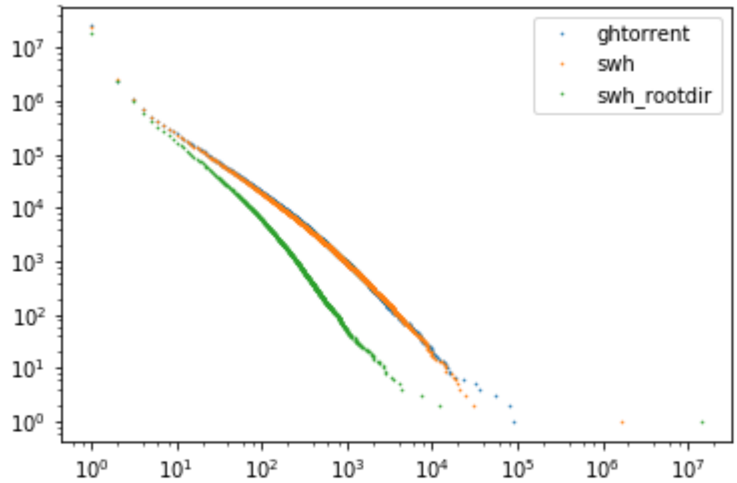
In [22]:
```
#Import des données
data={}
name=["ghtorrent","swh","swh_rootdir",]
for s in name:
    data[s]={}
    for ss in ["raw","cum"]:
        data[s][ss]={}
        with open(s+"_"+ss+".txt","r") as f:
            for line in f:
                x,y=line.split()
                data[s][ss][int(x)]=int(y)
        f.close()
#print(data)
```

In [9]:
```
# check global properties
no=0
for s in name:
    y_sum=0
    xy_sum=0
    for x,y in data[s]["raw"].items():
        y_sum+=y
        xy_sum+=x*y
    print(s,"# components",y_sum,"# origins",xy_sum)
    no=max(no,xy_sum)
print("ok if # of origins are =",no)
```

```
ghtorrent # components 25309069 # origins 41451739
swh # components 24017112 # origins 41451739
swh_rootdir # components 18536077 # origins 41451739
ok if # of origins are = 41451739
```

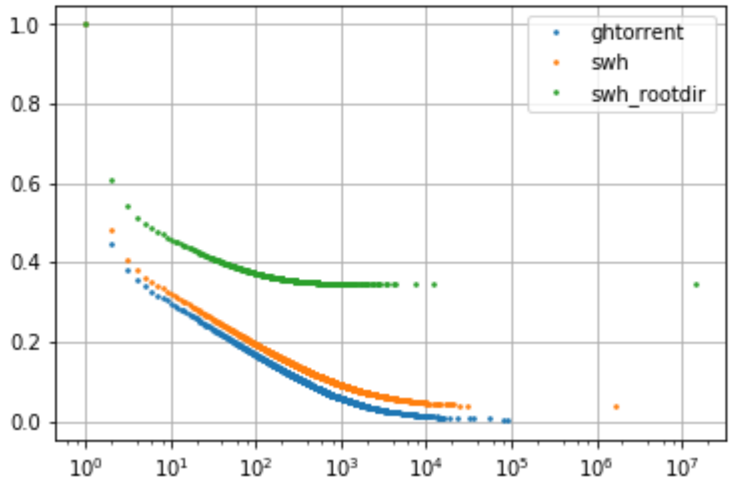
In [33]:
```
# affichage distribution cumulée des tailles des composants
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["cum"].keys()):
        x.append(key)
        y.append(data[s]["cum"][key])
    plt.loglog(x,y,"o",markersize=0.6,label=s)
    print(x[0],y[0])
plt.legend()
plt.show()
```

```
1 25309069
1 24017112
1 18536077
```



In [11]:
```
# affichage distribution cumulée des tailles des composants * la taille (ie le nombre d'origines dans des composants de tailles)
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum=np.zeros(len(y))
    # distribution cumulée de base
    for i in range(len(yysum)):
        yysum[i]=1*yy[i:].sum() # distribution non pondérée (identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
    plt.semilogx(xx,yysum/no,"o",markersize=1.6,label=s)
    print(xx[0],yysum[0])
plt.grid()
plt.legend()
plt.show()
```

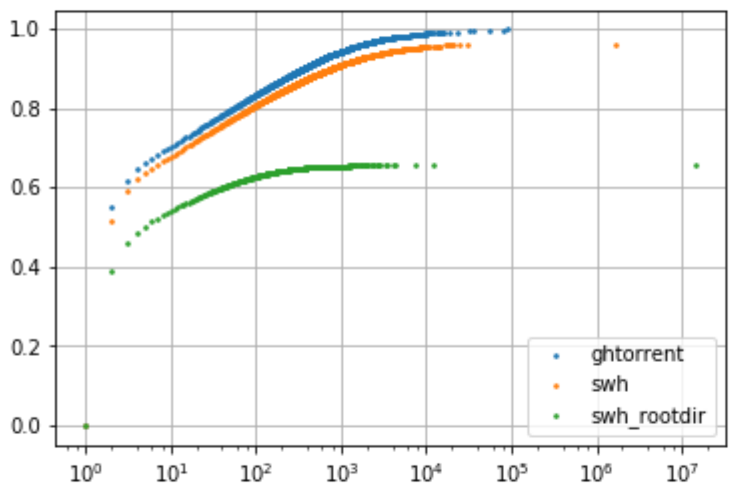
```
1 41451739.0
1 41451739.0
1 41451739.0
```



In [12]:

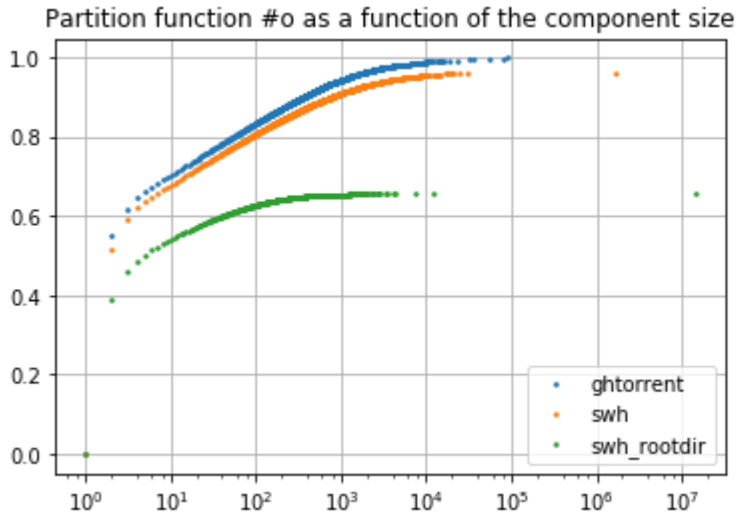
```
# affichage distribution cumulée des tailles des composants *
la taille (ie le nombre d'origines dans des composants de tai
lles)
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum=np.zeros(len(y))
    # distribution cumulée de base
    for i in range(len(yysum)):
        yysum[i]=1*yy[i:].sum() # distribution non pondérée
(identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
    plt.semilogx(xx,1-yysum/no,"o",markersize=1.6,label=s)
    print(xx[0],yysum[0])
plt.grid()
plt.legend()
plt.show()
```

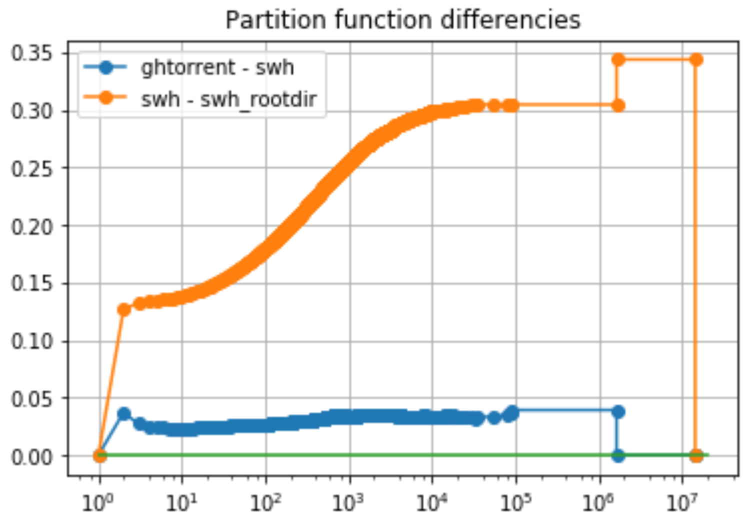
```
1 41451739.0
1 41451739.0
1 41451739.0
```

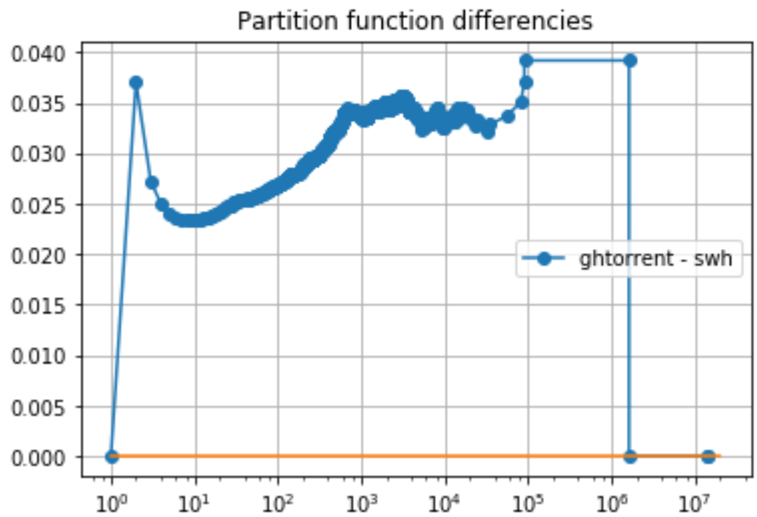


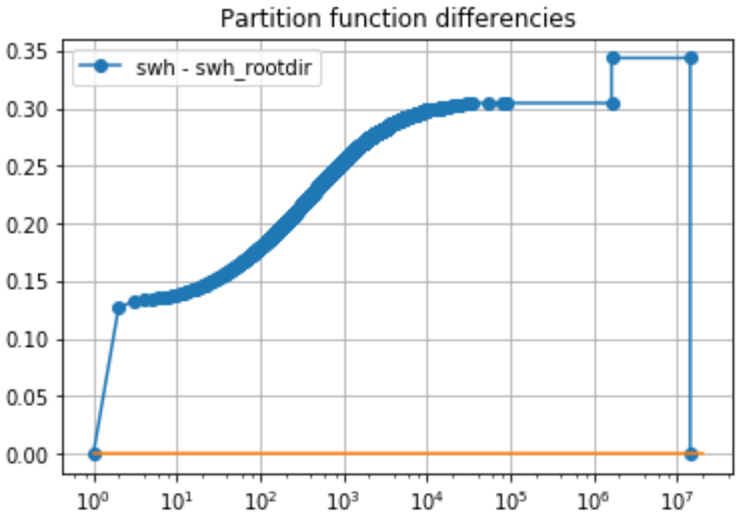
In [13]:
```
# on complete les histogrammes cumulées
#On crée une liste de toutes les valeurs de x
#puis on complete
# affichage distribution cumulée des tailles des composants *
la taille (ie le nombre d'origines dans des composants de tai
lles)
yysum={}
xxsum={}
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum[s]=np.zeros(len(y))
    xxsum[s]=xx
    # distribution cumulée de base
    for i in range(len(yysum[s])):
        yysum[s][i]=1*yy[i:].sum() # distribution non pondéré
e (identique à cum)
        yysum[s][i]=1-(xx[i:]*yy[i:]).sum()/no # distribution
pondérée
    plt.semilogx(xxsum[s],yysum[s],"o",markersize=1.6,label=
s)
    print(xx[0],yysum[s][0])
plt.grid()
plt.legend()
plt.title("Partition function #o as a function of the compone
nt size")
plt.show()
# liste des x complete
xxsum2={}
yysum2={}
for s in name:
    for i in range(len(xxsum[s])):
        x=xxsum[s][i]
        y=yysum[s][i]
        try:
            xxsum2[x][s]=1
        except:
            xxsum2[x]={s:1}
        try:
            yysum2[x][s]=y
        except:
            yysum2[x]={s:y}
    try:
        xxsum2[x+1][s]=1
        yysum2[x+1][s]=1
    except:
        xxsum2[x+1]={s:1}
        yysum2[x+1]={s:1}

#print(yysum2)
#on bouche les trous
default={}
for s in name:
```

```
1 0.0
1 0.0
1 0.0
```


Partition function #o as a function of the component size
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
ghtorrent
swh
swh_rootdir

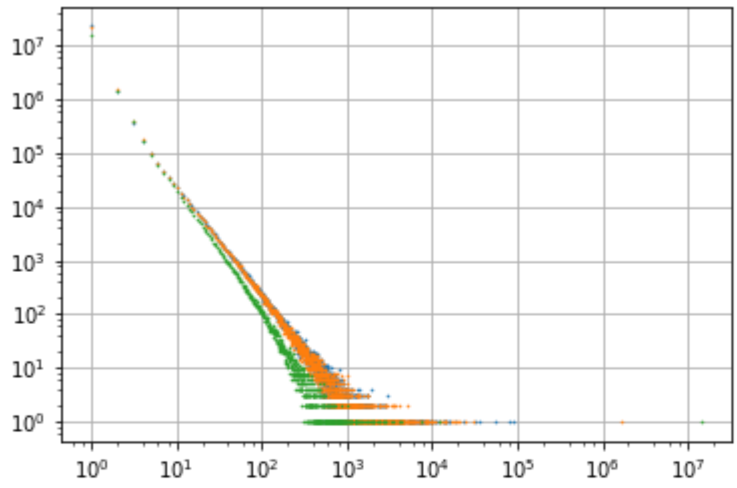

Partition function differencies
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
ghtorrent - swh
swh - swh_rootdir


Partition function differencies
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
ghtorrent - swh

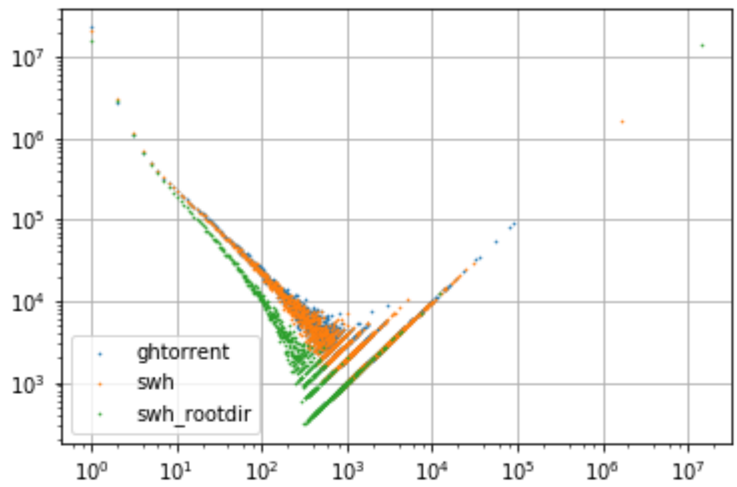

Partition function differencies
swh - swh_rootdir
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7

In [14]:
```
# affichage distributions BRUTES des tailles des composants
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    plt.loglog(x,y,"o",markersize=0.6)
    print(x[0],y[0])
plt.grid()
plt.show()
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key]*key)
    plt.loglog(x,y,"o",markersize=0.6,label=s)
    print(x[0],y[0])
plt.grid()
plt.legend()
plt.show()
```

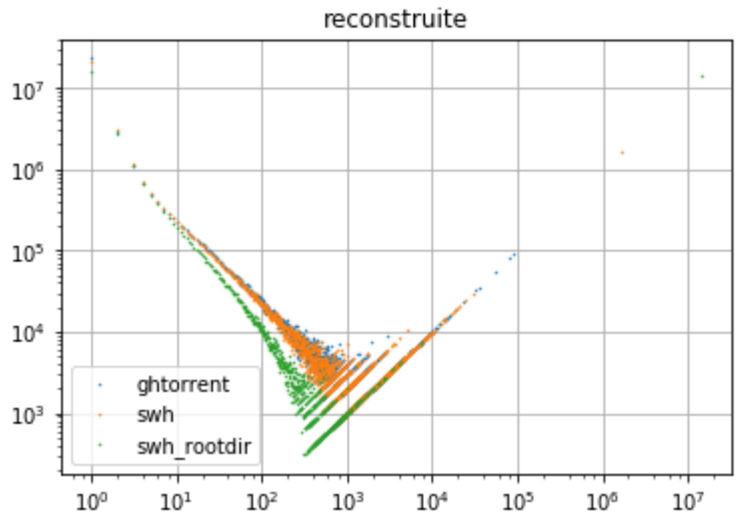
```
1 22906040
1 21372192
1 16131387
```




```
1 22906040
1 21372192
1 16131387
```

In [15]:
```
# distribution plot
for s in name:
    plt.loglog(xxsum3[0:len(xxsum3)-1],(yysum3[s][1:len(xxsum3)]-yysum3[s][0:len(xxsum3)-1])*no,"o",markersize=0.5,label=s)
plt.title("reconstruite")
plt.legend()
plt.grid()
plt.show()
```



In [16]:
```
print(xxsum3[len(xxsum3)-1])
print(xxsum3[len(xxsum3)-2])
```

```
14245628
14245627
```

In [17]:
```
for s in name:
    print(s,"-1",yysum3[s][len(xxsum3)-1])
    print(s,"-2",yysum3[s][len(xxsum3)-2])
```

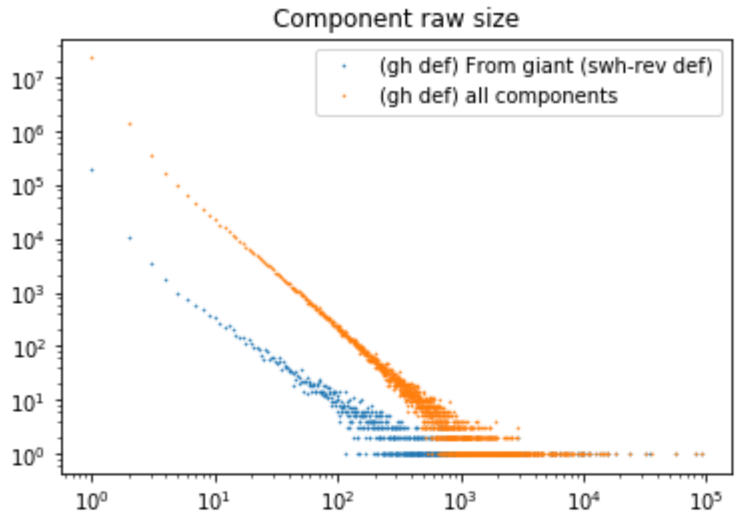
```
ghtorrent -1 1.0
ghtorrent -2 1.0
swh -1 1.0
swh -2 1.0
swh_rootdir -1 1.0
swh_rootdir -2 0.656332222877
```

lien avec la distribution des tailles au sens de la définition ghtorrent, dans l'amas géant au sens de swh (revision) 20200106

In [23]:
```
#Import des données
intersect={}
with open("intersect.txt","r") as f:
    for line in f:
        x,y=line.split()
        intersect[int(x)]=int(y)
f.close()
```

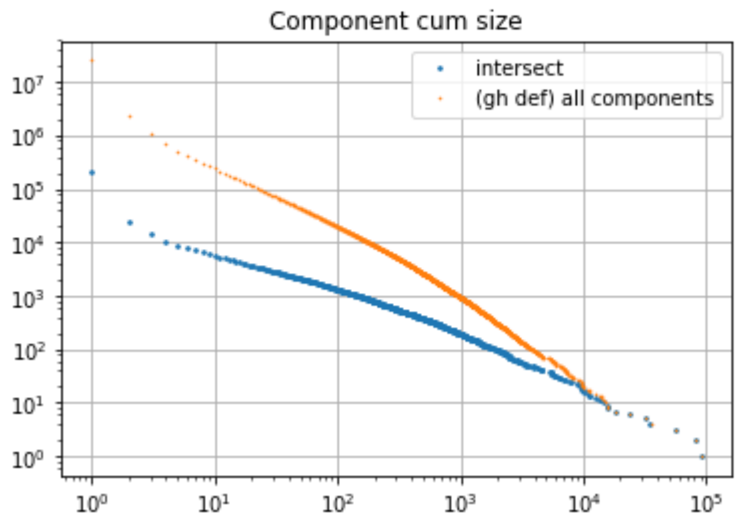
In [46]:
```
x=[]
y=[]
for key in sorted(intersect.keys()):
    x.append(key)
    y.append(intersect[key])
plt.loglog(x,y,"o",markersize=0.6,label="(gh def) From giant
(swh-rev def)")
print(x[0],y[0])
plt.grid()
#plt.show()
print(name)
for s in ["ghtorrent"]:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    plt.loglog(x,y,"o",markersize=0.6,label="(gh def) all com
ponents")
    print(x[0],y[0])
plt.grid()
plt.title("Component raw size")
plt.legend()
plt.show()
# show 1-Partition function (should be compared to "diff bewt
een partition function")
# check size of the giant component
y_sum=0
xy_sum=0
for x,y in intersect.items():
    y_sum+=y
    xy_sum+=x*y
print(s,"# components",y_sum,"# origins",xy_sum)
print("ok if # of origins are =",max(data["swh"]["cum"].keys
()))
```

```
1 190762
['ghtorrent', 'swh', 'swh_rootdir']
1 22906040
```



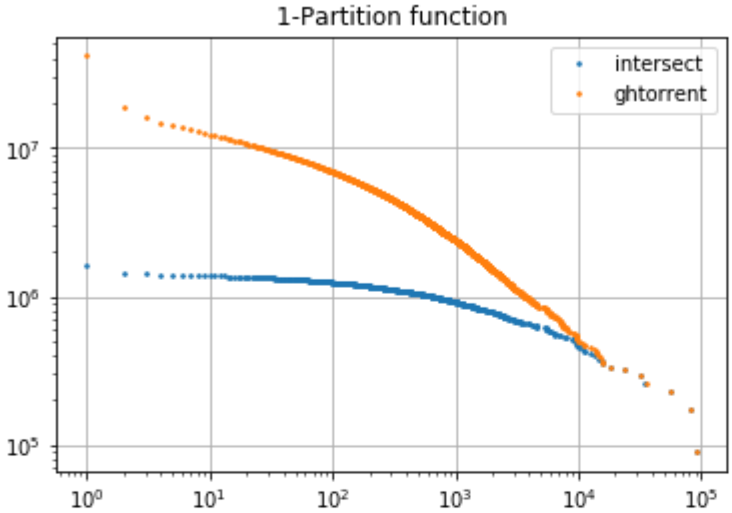

```
ghtorrent # components 215606 # origins 1624226
ok if # of origins are = 1624226
```

In [47]:
```
# affichage distribution cumulée des tailles des composants * la taille (ie le nombre d'origines dans des composants de tailles)
x=[]
y=[]
for key in sorted(intersect.keys()):
    x.append(key)
    y.append(intersect[key])
xx=np.array(x)
yy=np.array(y)
yysum=np.zeros(len(y))
# distribution cumulée de base
for i in range(len(yysum)):
        yysum[i]=1*yy[i:].sum() # distribution non pondérée (identique à cum)
        #yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
plt.loglog(xx,yysum,"o",markersize=1.6,label="intersect")

for s in ["ghtorrent"]:
    x=[]
    y=[]
    for key in sorted(data[s]["cum"].keys()):
        x.append(key)
        y.append(data[s]["cum"][key])
    plt.loglog(x,y,"o",markersize=0.6,label="(gh def) all components")
plt.grid()
plt.title("Component cum size")
plt.legend()
plt.show()
```

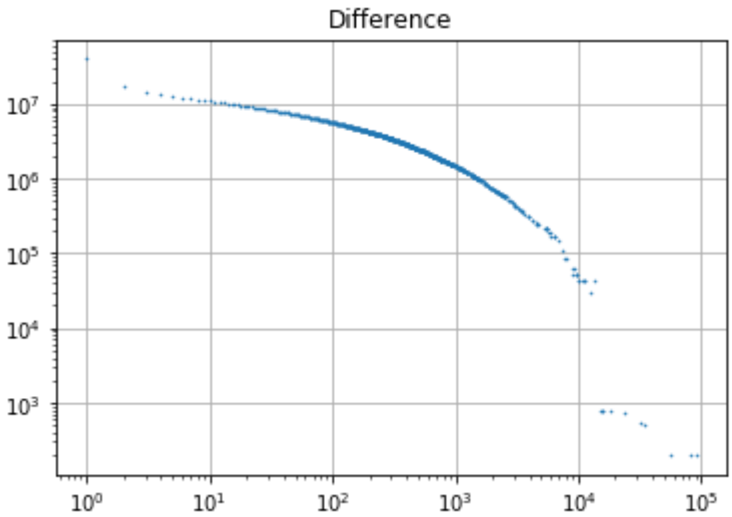


In [102]:
```
# Same with weighted, normalization, and diff
# affichage distribution pondérée cumulée des tailles des com
posants * la taille (ie le nombre d'origines dans des composa
nts de tailles)
x=[]
y=[]
for key in sorted(intersect.keys()):
    x.append(key)
    y.append(intersect[key])
xx=np.array(x)
yy=np.array(y)
yysum=np.zeros(len(y))
# distribution cumulée de base
for i in range(len(yysum)):
        #yysum[i]=1*yy[i:].sum() # distribution non pondérée
(identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
print(xx[0],yysum[0])
#no=yysum[0]
plt.loglog(xx,yysum,"o",markersize=1.6,label="intersect")

for s in ["ghtorrent"]:
    xref=[]
    yref=[]
    for key in sorted(data[s]["raw"].keys()):
        xref.append(key)
        yref.append(data[s]["raw"][key])
    xxref=np.array(xref)
    yyref=np.array(yref)
    yysumref=np.zeros(len(yref))
    # distribution cumulée de base
    for i in range(len(yysumref)):
        #yysumref[i]=1*yyref[i:].sum() # distribution non pon
dérée (identique à cum)
        yysumref[i]=(xref[i:]*yyref[i:]).sum() # distribution
pondérée
    plt.semilogx(xxref,yysumref,"o",markersize=1.6,label=s)
    print(xxref[0],yysumref[0])
plt.grid()
plt.title("1-Partition function")
plt.legend()
plt.show()


print(len(yysum),len(yysumref))
# diff where xxref exist
# xx ref value from intersect
# yysum truncated
xdiff=[]
ydiff=[]
for i in range(len(xx)):
    # find iref corresponding to xx[i] in xxref (ie xxref[ire
f]=xx[i])
    iref=np.where(xxref>=xx[i])[0].min()
    #print(i,iref[0].min())
    xdiff.append(xx[i])
    #ydiff.append((yysumref[iref]-yysum[i])/yysumref[iref])
    ydiff.append((yysumref[iref]-yysum[i]))
```

```
1 1624226.0
1 41451739.0
```

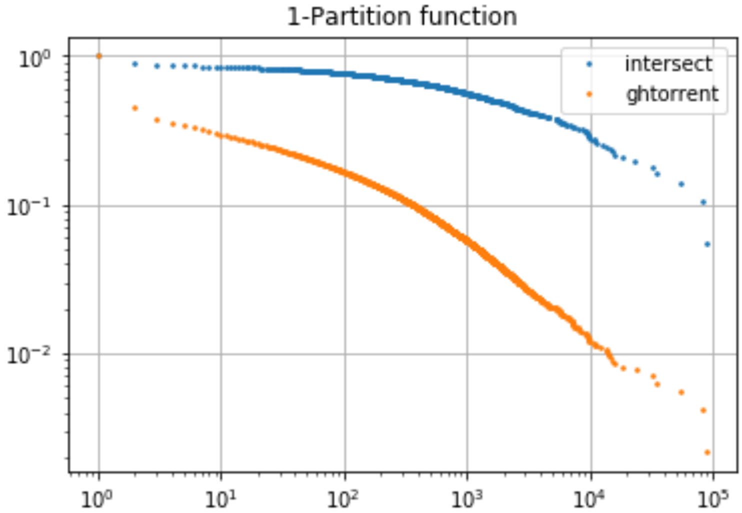



```
744 1684
```

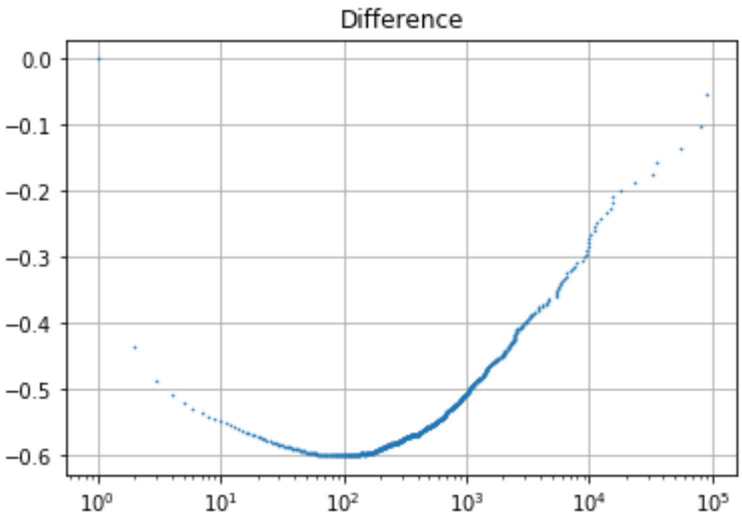


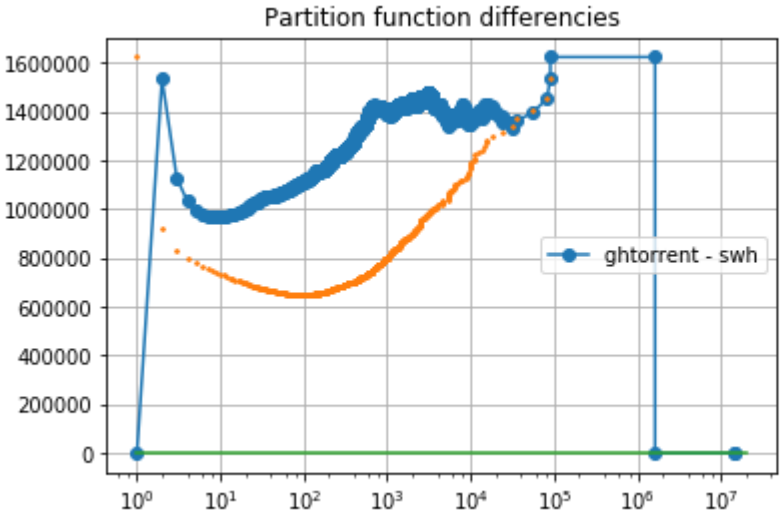
In [123]:
```
# Same with weighted, normalization, and diff
# affichage distribution pondérée cumulée des tailles des com
posants * la taille (ie le nombre d'origines dans des composa
nts de tailles)
x=[]
y=[]
for key in sorted(intersect.keys()):
    x.append(key)
    y.append(intersect[key])
xx=np.array(x)
yy=np.array(y)
yysum=np.zeros(len(y))
# distribution cumulée de base
for i in range(len(yysum)):
        #yysum[i]=1*yy[i:].sum() # distribution non pondérée
(identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
print(xx[0],yysum[0])
#no=yysum[0]
plt.loglog(xx/xx[0],yysum/yysum[0],"o",markersize=1.6,label="
intersect")

for s in ["ghtorrent"]:
    xref=[]
    yref=[]
    for key in sorted(data[s]["raw"].keys()):
        xref.append(key)
        yref.append(data[s]["raw"][key])
    xxref=np.array(xref)
    yyref=np.array(yref)
    yysumref=np.zeros(len(yref))
    # distribution cumulée de base
    for i in range(len(yysumref)):
        #yysumref[i]=1*yyref[i:].sum() # distribution non pon
dérée (identique à cum)
        yysumref[i]=(xref[i:]*yyref[i:]).sum() # distribution
pondérée
    plt.semilogx(xxref/xxref[0],yysumref/yysumref[0],"o",mark
ersize=1.6,label=s)
    print(xxref[0],yysumref[0])
plt.grid()
plt.title("1-Partition function")
plt.legend()
plt.show()


# coeff is ratio
ratio=yysum[0]/yysumref[0]

print(len(yysum),len(yysumref))
# diff where xxref exist
# xx ref value from intersect
# yysum truncated
xdiff=[]
ydiff=[]
for i in range(len(xx)):
    # find iref corresponding to xx[i] in xxref (ie xxref[ire
f]=xx[i])
```

```
1 1624226.0
1 41451739.0
```




```
744 1684
```

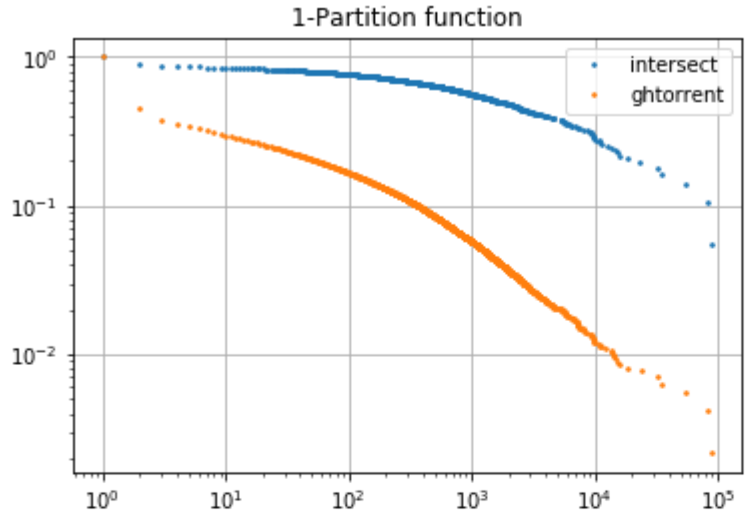




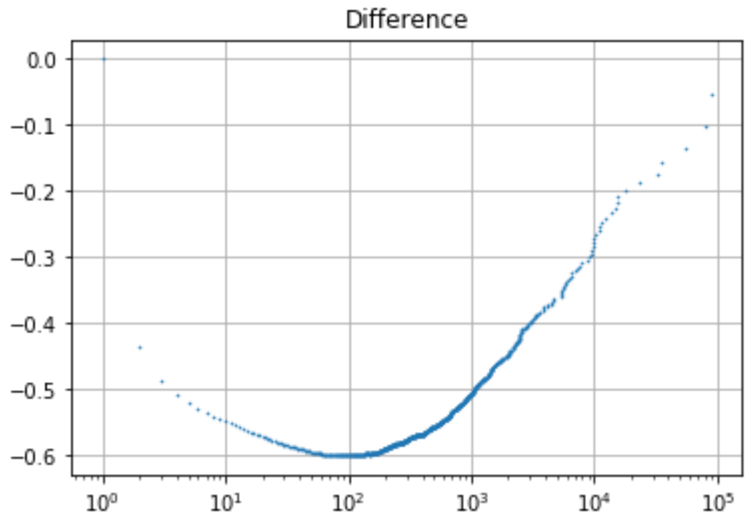
not yet convincing (still probleme in normalisation definition - solved in he next section)

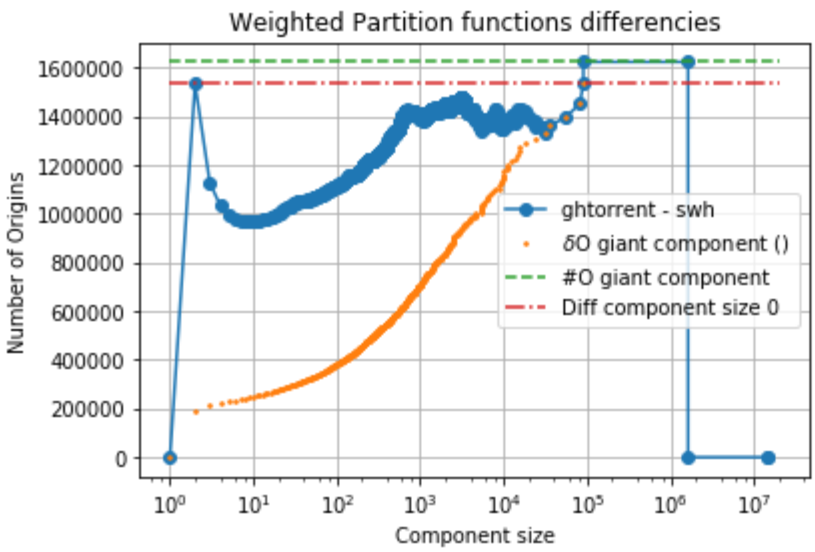
In [140]:
```
# Same with weighted, normalization, and diff
# affichage distribution pondérée cumulée des tailles des com
posants * la taille (ie le nombre d'origines dans des composa
nts de tailles)
x=[]
y=[]
for key in sorted(intersect.keys()):
    x.append(key)
    y.append(intersect[key])
xx=np.array(x)
yy=np.array(y)
yysum=np.zeros(len(y))
# distribution cumulée de base
for i in range(len(yysum)):
        #yysum[i]=1*yy[i:].sum() # distribution non pondérée
(identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
print(xx[0],yysum[0])
#no=yysum[0]
plt.loglog(xx/xx[0],yysum/yysum[0],"o",markersize=1.6,label="
intersect")

for s in ["ghtorrent"]:
    xref=[]
    yref=[]
    for key in sorted(data[s]["raw"].keys()):
        xref.append(key)
        yref.append(data[s]["raw"][key])
    xxref=np.array(xref)
    yyref=np.array(yref)
    yysumref=np.zeros(len(yref))
    # distribution cumulée de base
    for i in range(len(yysumref)):
        #yysumref[i]=1*yyref[i:].sum() # distribution non pon
dérée (identique à cum)
        yysumref[i]=(xref[i:]*yyref[i:]).sum() # distribution
pondérée
    plt.semilogx(xxref/xxref[0],yysumref/yysumref[0],"o",mark
ersize=1.6,label=s)
    print(xxref[0],yysumref[0])
plt.grid()
plt.title("1-Partition function")
plt.legend()
plt.show()


# coeff is ratio
ratio=yysum[0]/yysumref[0]

print(len(yysum),len(yysumref))
# diff where xxref exist
# xx ref value from intersect
# yysum truncated
xdiff=[]
ydiff=[]
for i in range(len(xx)):
    # find iref corresponding to xx[i] in xxref (ie xxref[ire
f]=xx[i])
```

```
1 1624226.0
1 41451739.0
```




```
744 1684
```





diff entre ghtorrent et swh pour les composants de taille 1 (1 533 848 origines) taille de l'amas géant 1624226

```
In [ ]:
```